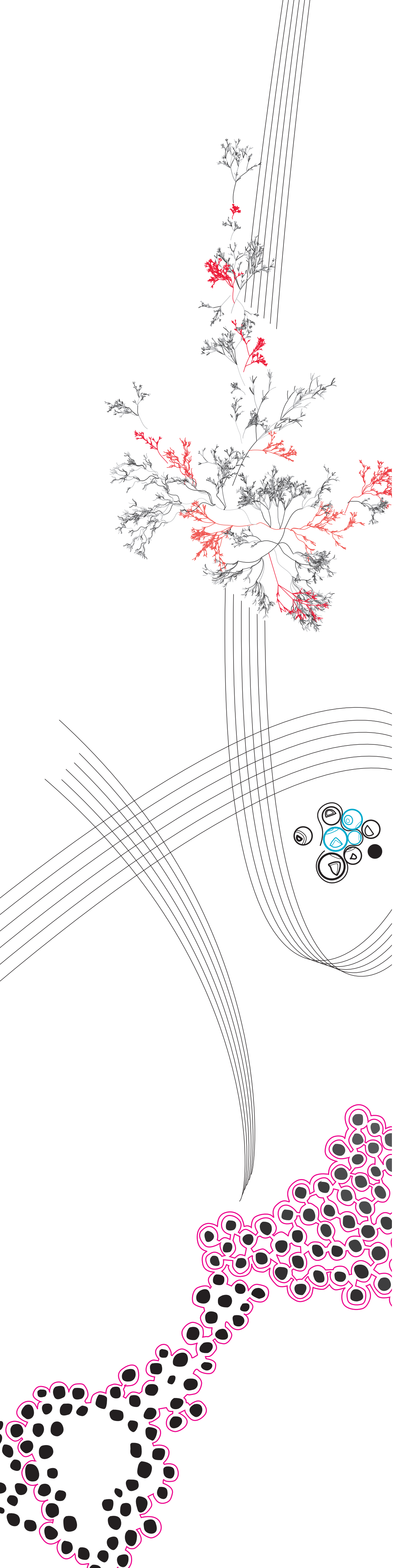


MASTER THESIS

Train Routing at the Shunt Yard: a Disjoint Paths Approach

Author:
Eline van Hove (s1314858)

Exam committee:
Joël van ´t Wout
Prof. Dr. Marc Uetz
Dr. Ir. Aleida Braaksma

February 2019

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), department of Applied Mathematics, chair Operations Research

UNIVERSITY OF TWENTE.

This thesis is the final product of my graduation project at NS (Nederlandse Spoorwegen), at the department Prestatieregie & Innovatie. *This 40 EC project is the last part of master Applied Mathematics at the University of Twente, the Netherlands.*

Abstract

This study considers the Shunt Routing Problem, which is to find routes (in terms of time and location) through the layout of a shunt yard. We model this problem as a Disjoint Paths Problem (DPP) on a time-expanded network. A path through this network corresponds to a route through the infrastructure of the shunt yard, in terms of time, location and direction. Additional constraints of a forbidden pair-form are needed to model the problem properly. Both, the DPP as well as the Impossible Pair constrained Path problem are NP-complete. We solve the problem using an ILP formulation and solver CPLEX. Results of experiments on twelve instances on stations Enschede and Eindhoven (the Netherlands) are promising, since the Disjoint Paths approach succeeds in finding schedules for significantly more train movements than the current algorithm does. Several heuristic methods are investigated to speed up the algorithm substantially, so that this new approach becomes applicable for practical implementation.

Keywords: shunt yard, Disjoint Paths Problem, time-expanded network, shunt routing, heuristics

Contents

List of Figures

Symbols and abbreviations

Abbreviation		**Introduced at page**
NS	Nederlandse Spoorwegen	6
OPG+	Opstelplan Generator	6
MIP	Mixed Integer Program	6
DPP	Disjoint Paths Problem	6
SSP	Successive Shortest Paths algorithm	6
RZ	RouteZoeker algorithm	6
BMI	Besturing Materieelinzet	10
GSSCG	Generator Spoor-Spoor Combinaties met Gewichten	10
TUSP	Train Unit Shunting Problem	10
DSP	Disjoint Shortest Paths algorithm	36
ILP	Integer Linear Program	22
MCUF	Multi-Commodity Unsplittable Flow problem	22
IPP	Impossible Pair constrained Path problem	31
DSP-C	Disjoint Shortest Paths algorithm for Composed movements	50

Symbol		**Introduced at page**
M (m)	Set of shunting movements	14
τ_{platform}	Platform track	14
τ_{park}	Park track	14
r_m	Release of movement m	14
d_m	Deadline of movement m	14
R (r)	Set of reservations of infra elements of the shunt yard	14
e	Infra element (track or switch)	14
(t_0, t_f)	Time interval of a reservation	14
M^S_{unp}	Set of unplanned movements in solution S	15
P_m	Route of movement m	15
$G = (V, E)$	Digraph, described by set of nodes and set of edges	22
$\mathscr{P}$	Set of paths in a graph	22
$V_{\text{source}}(v_m^{\text{source}})$	Set of source nodes	28
V_{A}	Set of nodes corresponding movements in A direction	28
V_{B}	Set of nodes corresponding movements in B direction	28
(s_i, t_i)	Source-target pair	28
$\delta \in \{A, B\}$	Driving direction	28
ω	Weight of an edge in the directed graph	28
E_{dummy}	Set of dummy edges for every source-target pair	28
$D_e \subseteq \{A, B\}$	Set of directions in which e can be approached	28
$x^m_{u,v}$	Decision variable which determines whether or not movement m is sent through edge $(u, v) \in E$	28
b^m_u	Node balance constraints	28
M^i (m^i_j)	Composed movement consisting of submovements	50

Introduction

The Dutch railway network is rather complex, observing the number of passengers, the dense railway infrastructure and the multiple actors involved in both passenger and freight transportation. The infrastructure network of approximately 7000 kilometer tracks in length contains over 400 stations (ProRail (2018)). The *Nederlandse Spoorwegen* (Netherlands Railways; NS) is the largest among several railway operators active in the Netherlands, transporting over 1.1 million passengers a day (ProRail (2018)). And this number of passengers is expected to grow the upcoming years (NOS (2018)).

Travelers depend on a safe, reliable and timely service. Providing this service requires careful planning of trains, crew and infrastructure. We distinguish multiple planning problems for passenger railways. This starts with *network planning*, focussing on the design of the network that will be used for operating the public transport services. This long term planning involves stations, yards as well as tracks. For NS, the planning starts with the *line planning* problem, determining origins, destinations and frequencies of lines in between. Choices strongly depend on the estimated demand. Assigning arrival and departure times to the services that will be operated on all railway lines is called *timetabling*. These times have to meet certain restrictions, such as travel times and dwell times at platforms to let people board and alight. *Rolling stock scheduling* assigns rolling stock to the timetabled services. Here one needs to take into account that there is a balance: for every planned departure, the appropriate train units need to be available at that location. The other way around: for every arriving train which is not departing immediately, there needs to be a location for storage. *Shunt planning* focuses on planning the local shunting processes at different stations, including the *maintenance planning* of rolling stock. Finally the *crew planning* combines the planned trips and activities into schedules for personnel.

In this study we focus on algorithmic decision support for railway planners who create operational plans for shunting processes at shunt yards. Specifically, our research considers the routing and scheduling of shunting movements at a shunt yard. Outside rush hours, an operator usually has a surplus of rolling stock. A *shunt yard* or *depot* is a storage facility, in most cases located close to a railway station, to park this surplus and execute service activities, such as cleaning, technical checks and repair activities. Around 40 stations in the Netherlands contain a depot.
Planning the operations on shunting yards is becoming the bottleneck in the planning of Dutch train infrastructure. Especially at night, when freight transportation takes over and most passenger trains are superfluous, shunting becomes a challenging puzzle. The capacity of shunt yards is limited and maintenance activities require facilities, crew and special type of tracks. Because of the growing number of train units, space and facilities become scarcer. Therefore there is a need for supporting planning tools to create feasible shunt plans.

In the current situation at NS, shunt plans are made by local planners without algorithmic support. A shunt plan describes the matching of arrival train units to departing train units, possibly merging or splitting trains. Furthermore the shunt plan describes a schedule of all shunt movements and routes. This problem is often referred to as the Train Unit Shunting Problem (TUSP). NS is developing a tool named OPG+ (Opstelplan Generator) to support shunt planners in finding a shunt plan. OPG+ splits the problem in multiple processes, which are solved sequentially using MIPs (Mixed Integer Programs) and the solver CPLEX. The planning of service activities such as cleaning, technical checks and repair activities are also incorporated in the tool. However, OPG+ does not always succeed in finding a shunt plan including all desired movements and service activities. In experiments performed on location Eindhoven, which is one of the biggest stations of the Netherlands, 10-15 percent of the planned movements could not be routed by OPG+. There is a need to improve the performance of the planning tool by decreasing the number of unplanned movements. That is the starting point for this study.

This study focuses on the problem solved by the last submodule of OPG+, RouteZoeker, which considers the routing problem at shunt yards. We name this problem the Shunt Routing Problem. Given a number of movements described by the start and end location and a time window in which this movement needs to be executed, the Shunt Routing Problem is to find for every movement a route through the layout of the shunt yard in terms of location and time. The primary goal is to minimize the number of unplanned movements.

We model the Shunt Routing Problem as a Disjoint Paths Problem (DPP) on a time-expanded network. In the time-expanded network, nodes correspond the infra elements at specific points in time in a specific direction. A path through the network corresponds to a route through the infrastructure of the station in terms of time and location. Additional constraints of a forbidden pair-form are needed to model the problem properly. Both, the DPP as well as the Impossible Pair constrainted Path problem (IPP) are NP-complete. We solve the problem using an ILP-formulation and solver CPLEX. Furthermore we present some heuristics to speed up the algorithm. We compare the results of our Disjoint Paths method to two other methods: the greedy Successive Shortest Path algorithm (SSP) and the method currently implemented in OPG+. For the experiments, we use a benchmark containing of 12 instances on station Enschede and Eindhoven. This benchmark is also used to investigate different heuristic methods to speed up the algorithm. Finally, to make a step towards practical implementation, we extend the algorithm for the Shunt Routing Problem. In this extended algorithm, we consider *composed movements*.

This report starts with an introduction in shunting to get familiar with the terms and to give a better understanding of the planning challenges. This chapter, Chapter 1 , describes the Train Unit Shunting Problem and its four subproblems: matching, parking, service scheduling and routing. Furthermore it describes the main layout of planning tool OPG+. Chapter 2 introduces the Shunt Routing Problem. In Chapter 3 we provide the reader with some background information about the scientific literature around shunting. The Disjoint Path formulation is introduced in Chapter 4, where we give a step-by-step construction of the underlying time-expanded network and a description of the ILP. This chapter also treats the complexity of the problem and discusses the time discretization which is needed to create the time-expanded network. In Chapter 5 the computational results of the experiments on the Disjoint Path algorithm are shown. To improve the running time of the algorithm, in Chapter 6 different heuristic methods are discussed. Chapter 7 describes the Composed Shunt Routing Problem and the changes needed to transform our Disjoint Paths algorithm to solve this extended problem. This chapter ends with the computational results of some exemplary experiments. This report ends with Chapter 8 discussing the conclusions of this study, the recommendations to NS and the suggestions for further research.

Chapter 1

Background information

Shunting involves parking and routing of rolling stock at shunt yards. During rush hours, roughly between 7:00 and 9:00 a.m. and 4:00 and 6:00 p.m., almost all rolling stock of passenger railway operators is in use. Outside these rush hours, there is a surplus of rolling stock. To fully exploit the main infrastructure for passenger and freight trains and to perform maintenance and cleaning activities, idle rolling stock needs to be parked at shunt yards. Especially during night time, most passenger trains are superfluous.

This chapter gives an introduction to shunting to get familiar with the terms and to give a better understanding of the planning challenges. Sections 1.1 and 1.2 form a brief introduction in the shunting jargon. In scientific literature, the planning problems concerning the shunting processes at a shunt yard are often referred to as the Train Unit Shunting Problem (TUSP). The TUSP and its four subproblems are introduced in Section 1.3. Section 1.4 describes the goals set by NS to improve the current planning processes at shunt yards and provides a brief description of OPG+, the tool developed by NS to support shunt planners which forms the starting point of this study.

1.1 Infrastructure of a shunt yard

Shunt planning starts with the topology of the considered shunt yard. In Figure 1.1 and Figure 1.2 two examples of Dutch shunt yards are depicted: station Zwolle and station Enschede. The platforms where people can board and alight are shown in green, service facilities are shown in red, yellow and blue. Enschede is a mid-size location and Zwolle is one of the most complicated shunt yards of the Netherlands.

Examples of characterizations of tracks are the following:

- The length of the track;
- The availability of catenary;
- The sides from which rolling stock can approach;
- The availability of several types of equipment;
- The availability of a railway safety system;
- The availability of a platform along the track.

We distinguish two configurations of a shunt track: a *stack* and a *deque*. The first type is open at one side and is often referred to as a *LIFO track* (Last In First Out). The latter is an open track which can be entered from both sides. The two sides are often referred to as the A- and B-side of the track, which

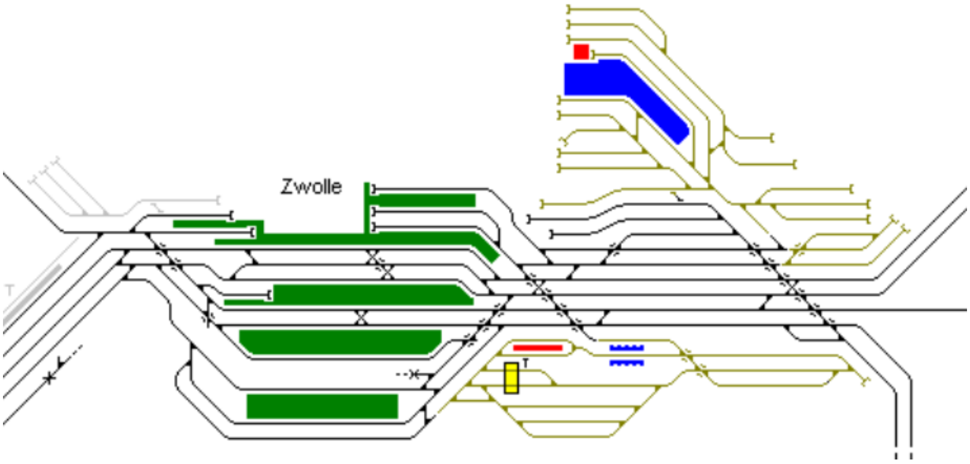


Figure 1.1: Infrastructure of station Zwolle.

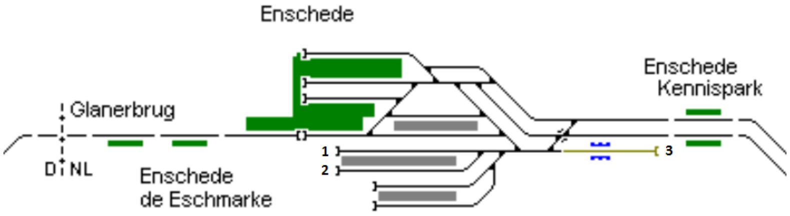


Figure 1.2: Infrastructure of station Enschede.

correspond to the A- and B-side of the station. In Figures 1.1 and 1.2 LIFO tracks are depicted with a little arc at the closed side.

Sawing

Consider tracks 1 and 2 at Figure 1.2 of station Enschede. As one can see, it is not possible to get from the track 1 to track 2 in one forward movement. To find a route, it is possible to use track 3 to change direction. This movement is called *sawing* and track 3 is referred to as a *saw track*. Sawing is an expensive operation in terms of time and crew. Therefore it is desired to minimize the number of sawing movements.

1.2 Conflicts

One of the important aspects in making a shunt plan is avoiding conflicts. This requires correct orderings of train units at parking tracks and strict time scheduling. One type of *conflict* occurs in a situation where two trains are routed (almost) simultaneously at the same track or switch.

Another conflict occurs at a shunt track whenever a train unit obstructs the arrival or departure of another train unit. These conflicts are caused by orderings at a track. Figure 2.1 illustrates this situation. In this figure the planned departure times are shown, the blocked train is depicted in gray and the planned directions are shown with arrows (where an arrow to both sides means that the train is allowed to leave in both directions). In the last example, switching the route by leaving the track via the B-side might be a solution to solve this last conflict (assuming the rest of the route does not cause new conflicts).

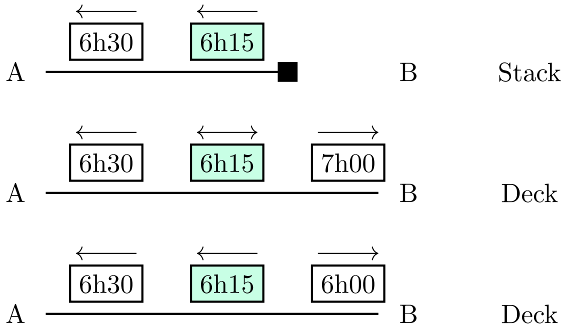


Figure 1.3: Three examples of conflicts due to orderings.

1.3 The Train Unit Shunting Problem

In the scientific literature, the planning problems concerning the shunting processes at a shunt yard are often referred to as the Train Unit Shunting Problem (TUSP). A first version of the TUSP was introduced by Freling et al. (2005). They consider the problem of parking train units overnight at a depot in such a way that each train unit can be retrieved, without moving others, when needed during the operations of the following day. Freling et al. (2005) decomposed the TUSP in two smaller subproblems: a matching problem and a parking (or track allocation) problem.

Lentink (2006) defined four subproblems of the TUSP. Besides the matching and parking problem, he formulated the routing and cleaning problem. We expand the latter one to maintenance tasks in general. We shortly explain these four subproblems:

- *Matching:* Given a timetable of arriving and departing train services, the Train Matching Problem is to find a feasible matching of arriving train units to departing train units, possibly (de-)coupling train units to fulfill the restrictions for the configuration of the train.
- *Parking:* Given a matching of arriving to departing train units, one needs to determine the location to store the resulting train units during the idle time in between arrival and departure. The Track Assignment Problem is to assign trains to shunt tracks in a feasible manner.
- *Service scheduling:* Given a set of assigned service tasks per train unit, such as internal or external cleaning, technical checks or repair activities, the Train Service Scheduling Problem is to find a feasible schedule which describes all planned service tasks in terms of time and location.
- *Routing:* After determining which train movements are needed between platform tracks, park tracks and service locations, unobstructed paths through the infrastructure need to be found.

N.B. one can regard crew planning as a fifth part of the TUSP. In this study, we focus on the fourth subproblem of the TUSP, routing. In Chapter 2 the full problem description can be found.

1.4 Shunt planning at NS

Shunt planning is part of the overall planning process of a railway system with passenger services. As described in the introduction, we distinguish also network planning, line planning, timetabling, rolling stock scheduling (of movements in between stations) and crew scheduling. Making the shunt plan is one of the latest steps in this overall planning process.

Nowadays, most of the shunt plans at NS are made by local planners without algorithmic support, using a pencil, paper and eraser. These planners are organized in different departments at different locations. For example, there is a big planning department located in Zwolle to create shunt plans for depots in the north and east of the Netherlands. In multiple time frames the planning is adapted at different levels of detail. Note that due to last minute disruptions, conflicts or crossings can occur in the shunt plan and replanning is desired. The planning of parking rolling stock and scheduling service activities are nowadays organized in parallel processes, at different departments. To improve the planning at depots and service sites, NS initialized a project called BMI ('Besturing Materieelinzet'). The main principles of this project are the following:

- *Short-term planning.* A planning is made for a 24-hour horizon, from rush hour to rush hour. To avoid rescheduling, the shunt plan is made using most recent information. Tools are developed to support shunt planners.
- *Integrality.* All rolling stock processes are integrally controlled. This means that there is one central responsibility for planning logistics, maintenance and service activities at a depot.

NS has developed a tool named OPG ("Opstelplan Generator") to support shunt planners in finding a shunt plan. OPG splits this problem in multiple processes, which are solved sequentially using MIPs (Mixed Integer Programs) and solver CPLEX. This version does not incorporate the planning of service activities. To satisfy the second principle of BMI, *integrality*, it is necessary to integrate the service scheduling in this tool. NS is therefore developing the tool OPG+, which is an extension of OPG incorporating the planning of maintenance activities. This tool forms the starting point of this study. Figure 1.4 depicts the global layout of OPG+, where the colored parts are the submodules already existing in OPG. The pre- and post-processor are added in OPG+ to include the service scheduling.

The first submodule, GSSCG ('Generator Spoor-Spoor Combinaties met Gewichten'), provides information to other submodules about routing possibilities at the depot. GSSCG outputs for every combination platform track - parking track or vice versa a fixed route in between and a weight indicating the *appropriateness* of this route. The Pre-Processor schedules the service activities which can be performed on less than 50 percent of the tracks. This is done by a greedy method, planning the activity at the first available track. OPG Kern is the main part of the algorithm, solving the matching and parking problem (see Section 1.3). It outputs for every movement a location and a time window in which this movement needs to be planned. Before this exact moment is planned by RouteZoeker, the Post-Processor schedules all remaining service activities. RouteZoeker finalizes the shunt plan by choosing for every movement the exact timing and the side of the platform track where this movement departs or arrives, using the routes which are already fixed by GSSCG.

The green marked symbols in Figure 1.4 refer to following information flows:

a. Time horizon, tracks and their length, routes, properties rolling stock, planned arrivals and departures (time, track, train unit), reservations rolling stock at the location, parameters, norms (for

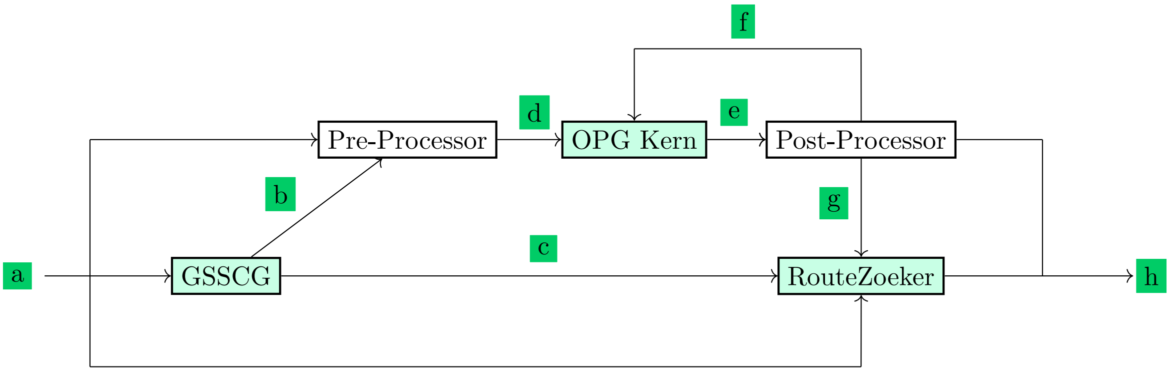


Figure 1.4: Global layout of OPG+. The green labels refer to the information flows.

example for combining and splitting), required service activities per train unit.

b. Weights for every combination platform track - parking track to indicate the *appropriateness* of this route.

c. For every combination platform track - parking track (or vice versa) a fixed route.

d. Planned service activities[1] (track, time), for every planned service activity an artificial arrival and departure.

e. Matching of the arrivals and departures, planned movements (time window, track, train unit), list of unplanned service activities.

f. Artificial arrivals and departures corresponding new planned service activities.

g. Planned movements (time window, track, train unit), planned service activities (time, track, train unit).

h. Planned movements (time, track including the side, train unit, routes), list of unplanned movements and activities.

[1] Only service activities which can be executed on the minority (less than 50 percent) of the tracks are planned in the Pre-Processor.

Chapter 2

The Shunt Routing Problem

The Shunt Routing Problem considers a routing and scheduling problem at a shunt yard at the end of the day, when trains are arriving at the shunt yard and need to be parked to stay overnight. Parking tracks are assigned upfront, as well as a time window in which this shunting movement from platform to parking track needs to be executed. When multiple trains are assigned the same parking track, the parking order is also determined to satisfy the order of departure the following morning. The task is to determine for every shunting movement a route through the infrastructure and a schedule which describes the execution times of those routes. Note that a similar problem arises at the beginning of the day, when trains parked at the shunt yard need to be routed to the platform tracks of departure. Furthermore, the problem can be generalized for movements between any pair of tracks at the railway station. For simplicity and readability, we restrict ourselves to the problem considering the planning problem at the end of the day. We define the Shunt Routing Problem as follows:

Shunt Routing Problem
Instance: We first list all relevant data and nomenclature, which is explained subsequently.

- The topology of the considered shunt yard, which consists of its tracks and switches.
- A set of shunting movements $M = \{m = (\tau_{\text{platform}}, \tau_{\text{park}}, k, r, d) \mid \tau_{\text{platform}} : \text{platform track}, \ \tau_{park} : \text{parking track}, \ k \in \mathbb{N} : \text{position}, \ r : \text{release}, \ d : \text{deadline}\}$, where k is an indicator for the position/order of the train at the parking track when multiple trains are assigned to parking track τ_{park}.
- A set of reservations $R = \{(e, t_0, t_f) \mid e : \text{ infra element (track/switch)}, \ t_0 : \text{start reservation}, \ t_f : \text{final time reservation}\}$, which determine the periods that certain infra elements (e) of the shunt yard are not available for shunting movements.

Goal: Finding a schedule where movements $m \in M$ are assigned an execution time and a route, satisfying the given time windows and assigned parking tracks and position, avoiding conflicts at the shunt yard and taking into account the reservations. The primary goal is to minimize the number of unplanned movements. A solution is feasible when all movements $m \in M$ are planned. Next to feasibility, the secondary objective is to minimize the costs.

The problem lives on the topology of the shunt yard, which is represented by its switches and tracks. In the definition, all movements $m \in M$ are denoted by 5-tuples $(\tau_{\text{platform}}, \tau_{\text{park}}, k, r, d)$. The platform track ($\tau_{\text{platform}}$) states where the train enters the shunt yard and the parking track (τ_{park}) states the assigned track to stay overnight. When this parking track is open at both sides, τ_{park} also includes the information at which of both sides (A or B) the train has to enter the park track. When multiple trains are assigned the same park track, an indicator k determines the order of the trains at the parking track, to take into account the order in which the trains must be retrieved when needed for operations the

following day. A time window ($[r, d]$) is given in which this movement needs to be planned. Here the release r denotes first possible time instant that a train is allowed to depart from its platform track. The deadline d denotes the last possible time instant that a train is allowed finish its movement.
A set of reservations, R, is introduced to take into account other planned activities at the shunt yard. A reservation can for example model a period that a track is out of use because of maintenance activities or it models a movement of other rolling stock through the infrastructure. These reservations are assumed to be fixed in time and place. For example, it is not possible to reroute a planned rolling stock activity or postpone the planned maintenance to a later moment in time.

The primary goal of the Shunt Routing Problem is to find a route and a schedule for the movements, minimizing the number of unplanned movements. When in the solution all movements are planned, we have a feasible solution. The secondary goal is to minimize the *cost*. This can be interpreted as cost in terms of time, equipment and crew. Let M^S_{unp} denote the set of unplanned movements in solution S. For a scheduled movement $m \in M \backslash M^S_{\text{unp}}$ let $n_m \in \mathbb{N}$ denote the number of infra elements on its route. The route of m is defined as a sequence of infra elements including the moments of arrival at these element: $P_m = ((e_1, t_1), (e_2, t_2), \ldots, (e_{n_m}, t_{n_m}))$ where e_i denotes the ith infra element on the route and t_i denotes the time of arrival at this element (for $i = 1, \ldots, n_m$). Here $(e_1, t_1) = (\tau_{\text{platform}}, r)$ and $e_{n_m} = \tau_{\text{park}}$. Note that it does not necessarily hold that $t_{n_m} = d$, since the train can arrive at the parking track before its deadline. A feasible schedule meets the restriction that consecutive elements e_i, e_{i+1} (for $i = 1, \ldots, n_m - 1$) in a route P_m are adjacent infra elements in the topology of the shunt yard. Furthermore, in assigning the planning times t_i, t_{i+1}, the minimum time needed to travel the ith infra element needs to be taken into account. Trivially, the schedule is restricted to usage of at most one train per infra element at a time. The parking tracks form an exception to this. When two movements m_1 and m_2 arrive from the same track and release $r_1 < r_2$, then movement m_1 needs to leave the track before r_2, the release of movement m_2. Finally, conflicts at crossings need to be avoided. In Figure 2.1 the layout of a crossing is depicted. Tracks 1 and 2 cannot be used simultaneously.

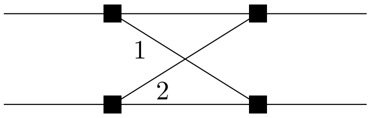


Figure 2.1: Layout of a crossing.

For simplicity and to clarify the scope of this project, the following assumptions are made:

- We assume the timetable and rolling stock schedule are given.
- We assume that the earliest departure takes place after the last arrival. This so-called *midnight constraint* has been introduced by Winter and Zimmermann (2000) and allows us to split up the problem and to consider the time horizon in the evening separately.
- We assume the shunt yard is empty at the beginning of the considered time horizon.
- We assume that the constraints imposed by the physical layout of the parking track are taken into account in the assignment of the parking tracks to the trains, for example the capacity of the parking track and the availability of catenary, are satisfied.
- The part of the parking track which is needed for parking the assigned trains is marked as "prohibited" to enter for other trains. Dependent of the capacity of the remaining part of the parking track we mark this part as "open for sawing movements" or not. (For the definition of *sawing* see Section 1.1.)
- We assume the length of the trains never exceeds the length of any track in the considered topology, including the parts of parking tracks which are marked open for sawing movements.
- (De-)coupling is outside the scope of this study.
- We assume the minimum time needed to travel an infra element is identical for all trains.

Some of these assumptions, for example the latter, can easily be adapted. However, we realize that these assumptions are limitations for implementation in real world. For example the assumption that the length of the trains never exceed the length of any track forms a simplification of reality. The input only describes simple movements from platform track to parking track to stay overnight. It might be desirable to incorporate routing via service locations at the shunt yard. In Chapter 7 we investigate the more complicated routing problem where composed movements are considered.

Chapter 3

Literature review

This chapter forms an overview of relevant literature and research on train shunting and specifically routing at shunt yards. Shunting is a broader problem than only within the passenger train infrastructure. In the scientific literature, shunting applications of tram scheduling (Winter (1999), Blasum et al. (2000), Winter and Zimmermann (2000)) and bus scheduling (Gallo and Di Miele (2001), Hamdouni, Desaulniers, and Soumis (2004)) can be found as well. For both buses and trams, the problems are similar to train shunting.

The Train Unit Shunting Problem

The planning problems concerning the shunting processes at a shunt yard are often referred to as the Train Unit Shunting Problem (TUSP). A first version of the TUSP was introduced by Freling et al. (2005). They consider the problem of parking train units overnight at a depot in such a way that each train unit can be retrieved, without moving others, when needed during the operations of the following day. Freling et al. (2005) decomposed the TUSP in two smaller subproblems: a matching problem and a parking (or track allocation) problem. Freling et al. (2005) formulated the matching problem as a mixed integer program (MIP) and solved using a commercial solver. The parking problem is modeled as a set partitioning problem with side constraints. It is solved using a column generation procedure.

Precise definitions of the TUSP differ per research, however we often see that the problem is decomposed in the matching and parking problems according to the definition of Freling et al. (2005). We observe that in most studies, service scheduling is not incorporated. A cost structure is often used to rank different solutions, however we also found papers where the TUSP is defined as a feasibility problem. Lentink (2006) continued the work of Freling et al. (2005), by formulating four instead of two subproblems: *matching, parking, routing* and *cleaning* (see Section 1.3). Lentink (2006) proves the NP-hardness of the matching and parking problems by reductions from the 3-Partition Problem and the Bin Packing Problem respectively.

Haijema, Duin, and Van Dijk (2006) report on a practical train shunting heuristic which shows similarities with Dynamic Programming. The matching and parking problems are again solved sequentially and isolated. Kroon, Lentink, and Schrijver (2008) are the first who propose a method to solve the two subproblems integrally. Based on Freling et al. (2005) they propose a large MIP formulation that solves the matching and parking problems simultaneously. The tool OPG+ developed at NS, is based on this principle: in a submodule of OPG+, OPG Kern, the matching and parking problem are solved integrally using a MIP formulation and solver CPLEX. Kroon, Lentink, and Schrijver (2008) distinguish tracks where multiple train types can be parked (*heterogeneous*) and where parking is restricted for only one train type (*homogeneous*). The concept of a virtual shunting track is introduced in order to help identify which tracks should be heterogenous and which tracks should be homogeneous from a unit type perspective. Depot tracks are used as homogeneous as possible. Routing and service scheduling are outside the scope of the research of Kroon, Lentink, and Schrijver (2008).

Den Hartog (2010) continued in his master thesis the work of Kroon, Lentink, and Schrijver (2008) using MIPs to solve the TUSP. He introduced a model to solve the matching, parking and routing integrally: the APT-model (Arrival on Park Track). The idea behind this method is that the quality of the solution can be improved by solving the subproblems integrally. However, this approach results in a huge amount of variables and constraints. To reduce the computation time, Den Hartog (2010) proposes two approximation methods to solve the APT-model: a one-stage and a two-stage solution method. The one-stage solution method reduced the number of variables and constraints by choosing the dwelling time (i.e. the time that a train is staying at the same (platform) track) as short as possible. However, this reduces the quality of the solutions. The two-stage solution method used a relaxed version of the one-stage solution method to propose a good order of train units on park tracks. Fixing this order, a second MIP is solved to create a feasible solution and to improve its quality. This study lacks an intensive comparison with other methods in terms of computational time and quality of the solution.

To the best of our knowledge, one of the earliest studies about shunting that aims at integrating the planning of maintenance activities is Jacobsen and Pisinger (2011). They consider the problem of scheduling the trains to workshops and depot tracks in order to complete the repairs as soon as possible, while avoiding train blockings at the tracks. Their research is motivated by a planning problem encountered by the Danish State Railway. Jacobsen and Pisinger (2011) present three heuristic approaches based on Guided Local Search and Simulated Annealing. However, their assumptions differ fundamentally from those of NS. They consider a time horizon of 2-3 days, working with discrete time units of one quarter. Furthermore they consider LIFO tracks only, assuming it is possible to collect any train unit at a depot track at any time within one quarter.

Haahr, Lusby, and Wagenaar (2017) introduce three new approaches for the TUSP: a constraint programming formulation, a column generation approach and a randomized greedy heuristic. They compare and benchmark these approaches with two existing methods: the MIP based on the paper Kroon, Lentink, and Schrijver (2008) and a two-stage heuristic based on the method of Freling et al. (2005). In the formulation of the TUSP by Haahr, Lusby, and Wagenaar (2017) only the subproblems *matching* and *parking* are considered, modelling any open-ended track as a LIFO track. Furthermore in Haahr, Lusby, and Wagenaar (2017) the TUSP is defined as a feasibility problem: instead of finding an optimal solution, the goal is to determine whether there exists a feasible solution given a rolling stock schedule. If there exists a feasible shunt plan, this solution is outputted.
In the computational experiments performed by Haahr, Lusby, and Wagenaar (2017), the column generation method was outperformed by the other methods. The randomized construction heuristic was able to solve almost all instances within one second. However, some harder and artificially generated instances were left unsolved by this method. In the benchmark, the two-stage method proved to be most successful, solving all but one of the feasible instances within a few seconds.

In OPG+, the TUSP is decomposed into different problems which are sequentially solved. Haahr, Lusby, and Wagenaar (2017) propose another decomposition method to cope with large instances: type and track decomposition. To reduce the solution space, they propose to decompose the problem instances by train unit types and tracks. A train unit type is restricted to park on a select subset of tracks. The partitioning of the tracks and train unit types can be performed such that the original problem decomposes into several independent problems, which can be solved individually. However, research is needed to determine an appropriate way to partition the tracks and train unit types.

Another extension to the TUSP is the TUSP with reallocation, proposed by Wolfhagen (2017) in her master thesis. Reallocations allow trains to switch park tracks multiple times during their stay at the shunt yard. Reallocation might be a method to improve the quality of the solution. However, incorporating reallocation in the MIP formulation results in many constraints and variables. Therefore Wolfhagen (2017) proposes a method which is a combination of row generation and tabu search. For smaller instances, this method outperformed OPG+. However, for larger sized problem instances, the computation time exceeded acceptable limits and no solution was found.

Another study which integrated the service scheduling and shunt planning is Van Den Broek (2016). He introduced in his master thesis at NS a local search based method. This algorithm was initially developed with a different purpose than OPG+, namely planning on tactical and strategical level. Different from other methods, Van de Broek regarded the service facilities as starting point. Infeasibility is avoided by allowing delay in the departure times. The goal is to minimize this delay caused by shunting processes. A planning is made starting from the available crew and equipment. The computational time and overall quality of the solutions are promising. However, in Van Den Broek (2016) some practical difficulties are not taken into account.

Routing

The problem of routing trains through railway stations usually occurs at three levels in the planning hierarchy of a railway company (Zwaneveld et al. (1996)). At the *strategic* planning level, the problem occurs in the analysis of future infrastructural capacity requirements, such as the number of platforms in the station and the number of tracks at the station yard. At the *tactical* planning level, the problem arises in the actual generation of timetables. Finally, at the *operational* level the problem occurs when timetables have to be adjusted for day-to-day disturbances, such as delays of trains. Zwaneveld et al. (1996) considered the problem of routing through railway stations from a strategical point of view. They formulate this problem as a feasibility problem and prove that this problem is NP-complete. Zwaneveld et al. present a model formulation based on the Node Packing Problem on conflict graphs. The vertices of a conflict graph represent the possible routes of the trains and two vertices are adjacent when planning both routes causes a conflict. A feasible solution to the routing problem is equivalent to an independent set in the associated conflict graph (i.e. there do not occur conflicts among the selected routes). However, this method does not give any information about the exact location of the conflict. Therefore, Hermann and Caimi (2006) continued this research by introducing a tree conflict graph, where for every possible train and starting time a route vertex is created and all possible routes in time are modeled separately. A vertex corresponds to a train at a topology element at a specific moment in time. Conflicts are modeled as undirected *conflict edges* in between pairs of nodes.

Tomii, Zhou, and Fukumara (1999) and Tomii and Zhou (2000) describe a genetic algorithm that handles storage of train units and several related processes, such as maintenance activities and cleaning. However, their shunting problem is of a less complex nature than the general shunting problem, since in their context at most one train unit can be parked on each shunting track at the same time.
Recent research about routing at railway stations is from Burggraeve (2017). Burggraeve solves the routing and timetabling problem sequentially, dividing the network in a bottleneck and the remainder of the network.

The problem studied by Van Den Broek (2009) comes closest to the problem we study. Van Den Broek introduces two MIP models for scheduling and routing shunting movements between shunting areas and platform areas of railway stations. Both models try to plan all shunting movements that have not been planned yet in between the already planned train movements. The goal is to verify whether it is possible to plan all these movements within the given time windows. In the first model, the routes of all movements are fixed, only the scheduling problem is solved. The second model allows selecting the route of a movement among a set of routes, which results in more feasible solutions. The models are tested for stations Utrecht, Zwolle and Groningen. For the latter two, the solution process takes too much computation time when solving the model with variable routes. The model with fixed routes is not able to plan the movements. Similar to Van Den Broek (2009), we observe that in most research about routing at shunt yards there is a predefined set of routes for every movement of which one itinerary needs to be chosen (see for example Bettinelli, Santini, and Vigo (2017), Burggraeve (2017) or Fuchsberger (2007)). This differs from our approach, where no routes are determined beforehand.

Time-expanded graphs

We will use time-expanded graphs to model our routing problem. Time-expanded graphs have already been used to model railway networks, for example in Cacchiani, Caprara, and Toth (2010) where it is applied to scheduling freight trains. Bettinelli, Santini, and Vigo (2017) construct time-space (di)graphs of trains separately to propose a fast algorithm for rescheduling trains. Their approach is based on the repeated execution of a greedy approach which schedules trains on a time-spaced network and several different dispatching rules. Computational testing has been performed on both real-world and generated instances. The obtained results show that the proposed algorithm is, within a few seconds, consistently capable of resolving existing conflicts and obtain high quality solutions. However, their approach also starts with a predefined set of possible routes per movement.
Fuchsberger (2007) considers the train scheduling problem via a resource constrained space-time integer multi-commodity flow in his master thesis. The space-time graph is constructed taking the tree conflict graph from Hermann and Caimi (2006) as starting point. The model has been tested succesfully on real-life instances of the Swiss Railways. Fuchsbergers approach does not take into account restrictions concerning the order of parking.

The order of parking is one of the aspects we want to incorporate in our model. Train blocks are parked sequentially on tracks, which limits the possibilities at departure. An order needs to be respected to avoid conflicts in the morning when the trains need to depart the parking track. This ordering problem is somehow similar to the container pre-marshalling problem studied by Lee and Hsu (2007). A container yard is considered, where containers are stacked high to utilize yard space more efficiently. The stacks of containers are comparable to the (LIFO) parking tracks. Containers need to be re-shuffled in such a way that it fits the loading sequence. In Lee and Hsu (2007) an optimization model is proposed, where the network embedded in the model is a multi-commodity flow problem based on a time-space network. The optimization goal is to minimize the number of container movements during the pre-marshalling. One of the differences between both problems is that every stack of containers consists of discrete 'places' of the same size, while the trains might differ in length.

Relation to our approach

The Shunt Routing Problem we defined is similar to the problem formulated by Van Den Broek (2009), where routing and scheduling of movements at a shunt yard are considered. Different from most earlier studies on routing at railway stations, we do not have a predefined set of routes for every movement. We will model the Shunt Routing Problem as a unsplittable network flow problem, using a time-expanded network. More specifically, we model the Shunt Routing Problem as a Disjoint Paths Problem with side constraints. Although time-space graphs have been used before to model railway networks, at the best of our knowledge, these networks are constructed differently from our approach. Lee and Hsu (2007) inspire us to incorporate restrictions due to the parking order at a track in the construction of the time-expanded graph.

Chapter 4

Disjoint Paths formulation

In this chapter, we model the Shunt Routing Problem as a Disjoint Paths Problem (DPP) with additional constraints and a minimization objective function. The DPP is a special type of the Multi-Commodity Unsplittable Flow problem (MCUF). Section 4.1 describes the construction of the time-expanded network as input graph of the DPP. Section 4.2 describes the ILP formulation and explains which additional constraints are needed to model the Shunt Routing Problem properly. The complexity of the problem is discussed in Section 4.3. This chapter ends with a section discussing the time discretization which is needed to construct the time-expanded network.

Multi-Commodity Unsplittable Flow problem (MCUF)
Instance: A directed graph $G = (V, E)$ where all edges are assigned a positive, integer weight denoting its capacity and a set K of 3-tuples $(s_1, t_1, d_1), \ldots, (s_k, t_k, d_k) \in V \times V \times \mathbb{N}$ where each 3-tuple (s_i, t_i, d_i) $(1 \leq i \leq k)$ denotes a commodity with d_i its demand, s_i its source node and t_i its target (or sink) node.
Goal: Determine a set of paths $\mathscr{P} = \{P_1, \ldots, P_k\}$ through the network of G, such that P_i accommodates demand d_i from s_i to t_i $(1 \leq i \leq k)$, such that the total amount of flow (i.e. flow of all commodities) through the edges does not exceed the capacities of the edges.

Disjoint Paths Problem (DPP)
Instance: A directed graph $G = (V, E)$ and a set K of 2-tuples $(s_1, t_1), \ldots, (s_k, t_k) \in V \times V$ where each 2-tuple (s_i, t_i) $(1 \leq i \leq k)$ denotes a source (s_i) and target (or sink) node (t_i).
Goal: Determine a set of node-disjoint paths $\mathscr{P} = \{P_1, \ldots, P_k\}$ through the network of G, such that P_i is a path from s_i to t_i $(1 \leq i \leq k)$.

In the MCUF different commodities need to be shipped simultaneously through a common network, such that the total flow going through each edge does not exceed its capacity. Note that $\mathscr{P}$ contains k paths, so commodities cannot be split to use different paths. The Disjoint Paths Problem is to find a set of node-disjoint paths in a network. If all capacities and demands equal one, the MCUF is equivalent to the edge-disjoint paths problem, which can be reduced to the DPP. The definition of the edge-disjoint paths problem equals the definition of the DPP, despite the word *node-disjoint* which is replaced by *edge-disjoint*. An instance to the DPP can easily be transformed to an instance of the edge-disjoint paths problem by duplicating all vertices and connecting them with an edge with capacity equal to one.

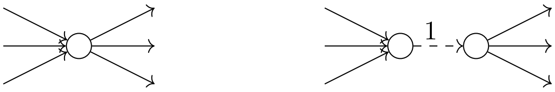


Figure 4.1: Duplication of nodes to transform an instance of the node-disjoint paths problem to an instance of the edge-disjoint paths problem.

The construction of such a duplication is shown in Figure 4.1. The other way around, the edge-disjoint paths problem can be reduced to the DPP by its line graph (see Schrijver (2003)).

The MCUF arises in a wide variety of application settings in communication, logistics, manufacturing and transportations (Jiao and Dong (2018)). For an NP-hardness proof for the MCUF see, for example, the textbook of Ahuja, Magnanti, and Orlin (1993). For the NP-hardness of the DPP (the directed as well as the undirected version) see Karp (1975). The problem remains NP-complete even if G is constrained to be planar (Lynch (1975)). The work of Robertson and Seymour (1995) says that there is a polynomial time algorithm (actually an $O(n^3)$ time algorithm) for the DPP when the number of terminals, k, is fixed. Kawarabayashi, Kobayashi, and Reed (2012) continue with this result and present an algorithm to solve the DPP in quadratic time, when is assumed that k is fixed. Although these results sound promising, their algorithm is not applicable in our study, since we need additional constraints to properly model the Shunt Routing Problem. Furthermore, this is a theoretical result where computational time might explode even for moderate values of k.

4.1 The time-expanded network

We will model the Shunt Routing Problem as a DPP with additional constraints. Given an instance to the Shunt Routing Problem: movements M, reservations R and the topology of the shunt yard, we describe in this section the construction of an instance to the DPP. We illustrate the steps of construction by a small example, of which the topology of the shunt yard is shown in Figure 4.2. We see three tracks, of which track 2 and 3 are parking tracks which can be used as sawing tracks (definition see Section 1.1). In this figure, α denotes a switch. The gray arrows denote the two directions in which a train can move at the shunt yard. In this picture the left side is referred to as the A-side and the right as the B-side of the shunt yard.

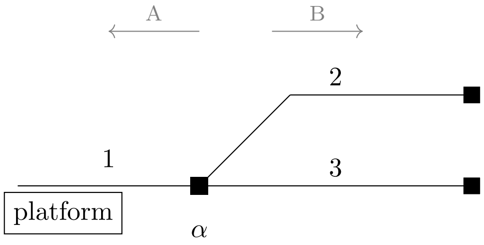


Figure 4.2: Small example of the topology of a shunt yard.

1. **Preprocessing**
   a. Split open parking tracks in two LIFO tracks, one open at side A and one open at side B. Both 'parts' are treated as independent LIFO parking tracks. (Note that this is possible because departures from parking tracks are not considered. Otherwise it is possible that a train enters an open track at side A and leaves at side B or vice versa.) The place where to cut the track can be deduced by the movements assigned to this parking tracks, since the side at which they enter the track is given. Figure 4.3 depicts an example, where arrows indicate the direction of approach of the trains assigned to the open parking track. The dashed line shows where to cut the track in two

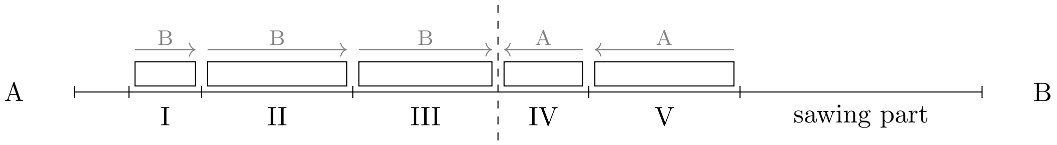


Figure 4.3: Example of an open parking track decomposed in two LIFO tracks, which are again decomposed in a sawing part and parts for the parking of every assigned train.

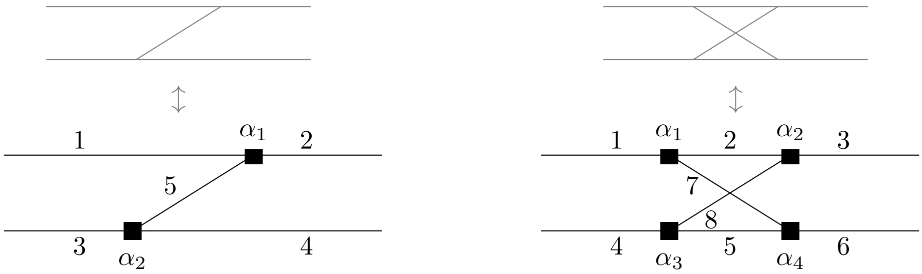


Figure 4.4: Decomposition of crossings (shown in gray) in track segments and switches.

LIFO tracks.
b. Determine for every LIFO parking track whether there exists a sawing part. Note that, since we know where trains must be parked, we can determine the situation at a parking track at the end of the considered time horizon, assuming all movements are executed. If there is *enough* idle capacity of the parking track in this final situation, this idle part is treated as independent track which is open for sawing movements. In the example of Figure 4.3 a part is marked as *sawing part* at the right part of the parking track. From now on, this part is considered as an individual track segment.
c. Split parking tracks in parts for every train which is planned to be parked at the track. In Figure 4.3 these parts are denoted by roman numerals. Every part is modeled as an individual track segment.
d. Decompose crossings in track segments and switches according to the example in Figure 4.4. In the left crossing it is for example possible to drive from track part 3 via 5 to 2 or from track part 3 to 4.

2. **Construction of the topology network**
Model every track segment or switch ('infra element') as two nodes for the two directions in which a train can approach the element. For example, a vertex '1B' denotes track 1, driving in the B-direction. Add edges corresponding the possible movements between these elements. Assign weights to the edges indicating the time needed to travel the specific element. The network in Figure 4.5 is constructed from the example layout shown in Figure 4.2. Sawing movements (i.e. changing direction at the same track) are depicted in green. We assume in this example that sawing takes 2 time units and traversing switches or tracks 1 time unit. Let us consider the movement depicted in bold: the train starts at track 1, moving in the B-direction. It enters switch α in direction B and enters track 3 to change direction. Let us refer to the resulting network as *topology network*.

3. **Construction of the set of vertices**

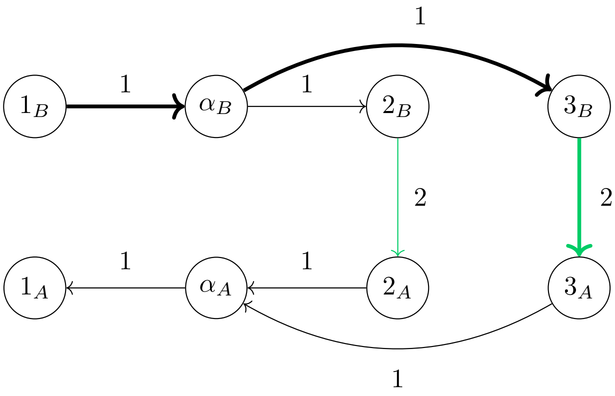


Figure 4.5: *Topology network* of the shunt yard shown in Figure 4.2.

From the topology network, we create a (weighted) time-expanded graph $G = (V, E)$. We duplicate the nodes of the topology network for every time step for the chosen granularity. Vertices correspond to an infra element, a direction of approach and a moment in time. Note that in the preprocessing, parking tracks might be split in several infra elements. V_A denotes the set of vertices corresponding elements approached in the A direction and V_B denotes the set of vertices corresponding to elements approached in the B direction. (Note that if an infra element is approached in the A direction, it enters the element at its B-side and vice versa. This might be a bit confusing.) Furthermore we create dummy nodes, V_source, for every movement which departs from a platform track which is open at both sides. We define the set of vertices as follows:

$$V = V_\text{A} \cup V_\text{B} \cup V_\text{source},$$

where V_source is defined below and

$$\begin{aligned} V_\text{A} &= \{v_{e,t,A} \mid e : \text{ infra element which can be approached in the A direction, } t : \text{time instant}\}; \\ V_\text{B} &= \{v_{e,t,B} \mid e : \text{ infra element which can be approached in the B direction, } t : \text{time instant}\}. \end{aligned}$$

4. **Marking sources and targets**
We need to define $(s_1, t_1), \dots (s_k, t_k) \in V \times V$ to define source and target nodes between which paths need to be found. A path from source to target models a route through the topology in terms of location and time. For every movement $m = (\tau_\text{platform}, \tau_\text{park}, k, r, d) \in M$ we mark one vertex in V as source (s_m) and one vertex as target (t_m). If τ_platform is open at two sides, we create a dummy node and connect this node with the two nodes corresponding both sides of the platform track. The set of these dummy source nodes is defined as

$$V_\text{source} = \{v_m^\text{source} \mid m : \text{movement starting at platform track opened at both sides}\}.$$

We mark all these dummy nodes as source nodes, i.e. $s_m = v_m^\text{source}$. If τ_platform is a LIFO track which can be approached in the direction $\delta \in \{A, B\}$, we set $s_m = v_{\tau_\text{platform}, r, \delta}$. Note that r corresponds to the release of the movement. As target we set $t_m = v_{\tau_{\text{park},k}, d, \delta}$, where $\tau_{\text{park},k}$ denotes the parking track at position k, d denotes the deadline and δ denotes the direction in which this movement enters its parking track.

5. **Construction of the set of edges**
We construct the set of edges in the time-expanded network, E, with weights $\omega : E \to \mathbb{N}$, such that a path in the graph corresponds to a movement in terms of topology and time. We distinguish several types of edges. We list them below.

- *Dummy edges* (E_dummy) are constructed between pairs (s_m, t_m) for all $m \in M$, to model unplanned movements. These edges are assigned (high) weights $\omega_m^\text{unplanned}$ to penalize unplanned movements.
- *Waiting edges* (E_wait) model waiting at the infrastructure. This is only possible at tracks. Waiting at a track τ is penalized by weight $\omega_\tau^\text{wait} \in \mathbb{N}$.
- *Movement edges* (E_move) model movements between two different infra elements. Length or cost to move from element i to j is modeled as $\omega_{i,j}^\text{move} \in \mathbb{N}$.
- *Sawing edges* (E_saw) are movements on the same track changing direction. Weights $\omega_\tau^\text{saw} \in \mathbb{N}$ are introduced to penalize using track τ for sawing movements.
- *Source edges* (E_source) are added for movements from open platform tracks. For such a movement m, two *source edges* are added between v_m^source and the nodes corresponding both directions of the platform track. These edges have weight zero.

Let $D_e \subseteq \{A, B\}$ denote the set of directions in which infra element e can be approached.

$$
\begin{aligned}
E =& E_{\text{wait}} \cup E_{\text{move}} \cup E_{\text{saw}} \cup E_{\text{source}} \cup E_{\text{dummy}}, \\
\\
E_{\text{wait}} =& \{(v_{e,t,\delta}, v_{e,(t+1),\delta}) \mid \delta \in D_e,\ e:\ \text{track element},\ t: \text{time instant}\}; \\
E_{\text{move}} =& \{(v_{e_1,t,\delta}, v_{e_2,(t+t_{1,2}),\delta}) \mid \delta \in D_{e_1} \cap D_{e_2},\ e_1, e_2:\ \text{adjacent infra elements when approach-} \\
& \text{ing in direction } \delta,\ t: \text{time instant},\ t_{1,2}: \text{duration of a movement from } e_1 \text{ to } e_2\}; \\
E_{\text{saw}} =& \{(v_{e,t,\delta_1}, v_{e,(t+t_{\text{saw}}),\delta_2}) \mid \delta_1, \delta_2 \in \{A, B\},\ \delta_1 \neq \delta_2,\ e:\ \text{sawing track},\ t_{\text{saw}}: \text{duration of a} \\
& \text{sawing movement},\ t: \text{time instant}\}; \\
E_{\text{source}} =& \{(s_m, v_{\tau_{\text{platform}},r,\delta}) \mid m \in M, \delta \in D_{\tau_{\text{platform}}},\ \tau_{\text{platform}}:\ \text{platform track of movement } m, \\
& s_m:\ \text{source node of } m,\ r: \text{release of } m\}; \\
E_{\text{dummy}} =& \{(s_m, t_m) \mid m \in M,\ s_m:\ \text{source node of } m,\ t_m:\ \text{target node of } m\}.
\end{aligned}
$$

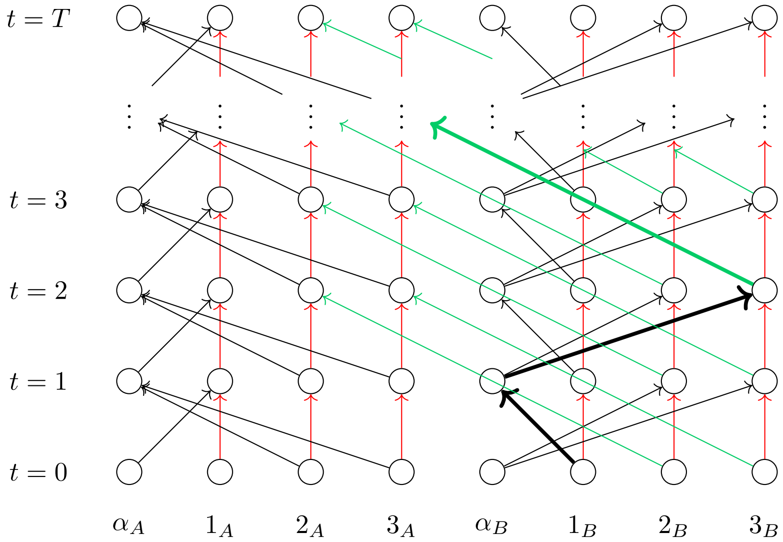
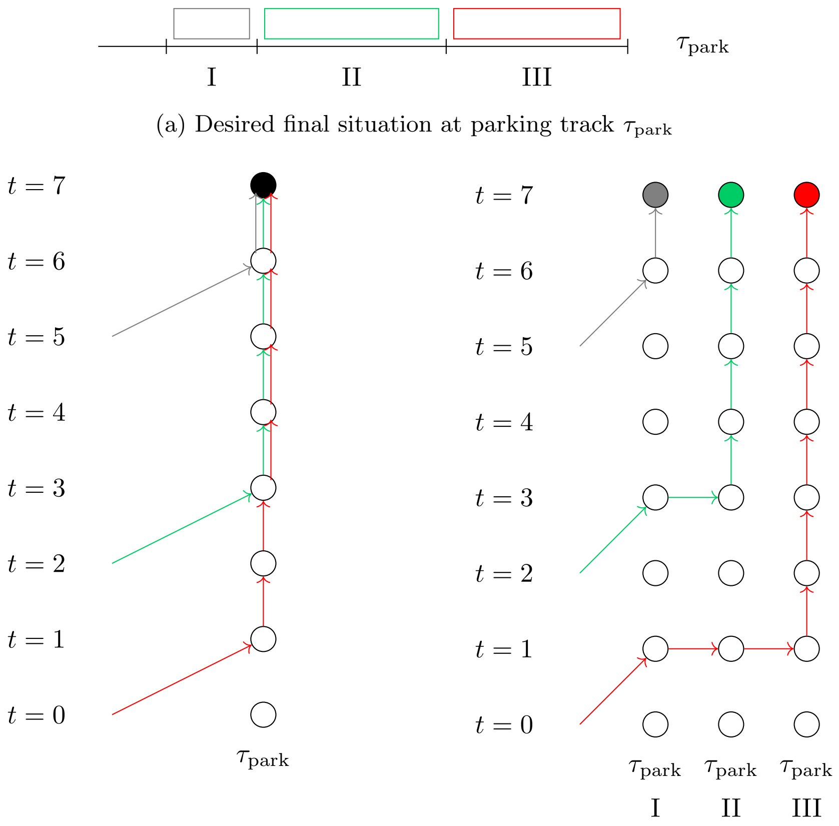


Figure 4.6: Time expanded network based on the graph in Figure 4.5. The movement depicted in bold corresponds to the one in Figure 4.5 starting at time $t = 0$.

In Figure 4.6 the time expanded network of Figure 4.5 is depicted for T time steps. The *sawing edges* are depicted in green and the *waiting edges* in red. The movements depicted in bold in Figures 4.5 and 4.6 correspond, assuming the movement starts at time $t = 0$.

Figure 4.7 illustrates why in the first step of the construction, parking tracks are split in components for every assigned train. Without splitting the parking track, all trains (depicted in gray, green and red) would have had the same target node. This situation would have forced the movements to use the same nodes (and edges). By splitting the parking track in segments, this situation is avoided: the three trains can arrive at their destination using disjoint paths. When driving to the k^{th} part of the track, a train need to pass the previous $k - 1$ parts at a moment that all those parts are empty. If there is a non-empty track, a path of waiting arcs in the network blocks paths to further track parts. Therefore, finding disjoint paths for trains to all parts is only possible when the trains enter the park track in the correct order.

6. **Removal of reservations R**
   To take into account the reservations R, we need to remove vertices and edges from the graph. Let (e, t_0, t_f) be a reservation of element e in the time interval $[t_0, t_f]$. We remove all nodes and adjacent edges corresponding this element in between the idle time interval to avoid planning movements at these tracks and switches.



(a) Desired final situation at parking track τ_{park}

(b) Network layout with and without segmentation of the parking track

Figure 4.7: Example illustrating the segmentation of parking tracks to model the position on parking track τ_{park}.

The resulting graph $G = (V, E)$ is an instance of the Disjoint Paths Problem where node-disjoint paths need to be found between the pairs $(s_m, t_m) \in V \times V$ for all movements $m \in M$. A path between the source (s_m) and target (t_m) corresponds to a route and execution time for movement m. Note that it is possible that the train departs later than its release time r, using waiting edges, which will block the considered track during this waiting period. Equivalently, a train can arrive earlier than the deadline d using waiting edges at the end of the path.

4.2 ILP formulation

We can model the Shunt Routing Problem as a Disjoint Paths Problem on the graph $G = (V, E)$ constructed in Section 4.1 with additional constraints. In this section we present the ILP formulation of this model and we explain which additional constraints are needed to properly model the Shunt Routing Problem.

Note that we need to duplicate all nodes in V according to the construction shown in Figure 4.1 to transform the instance of the node-disjoint paths problem to an instance of the edge-disjoint paths problem. In this formulation $\omega_{u,v}$ denotes the weight assigned to edge (u, v). Note that this edge contains information about the infra elements, the direction of the movement and the time. The decision variables are defined as follows:

$$x^m_{u,v} = \begin{cases} 1, & \text{if movement } m \text{ is sent through edge } (u,v) \in E \\ 0, & \text{otherwise.} \end{cases}$$

Now the ILP for the Shunt Routing Problem can be formulated as:

$$\begin{aligned}
\min_x \quad & \sum_{(u,v)\in E} \sum_{m\in M} \omega_{u,v} x^m_{u,v} && && (4.1)\\
& \sum_{m\in M} x^m_{u,v} \leq 1 && \forall (u,v) \in E && (4.2)\\
\sum_{(u,v)\in E} x^m_{u,v} - & \sum_{(v,u)\in E} x^m_{v,u} = b^m_u && \forall u \in N, \forall m \in M && (4.3)\\
& (4.4) - (4.8) && &&\\
& x^m_{u,v} \in \{0,1\} && \forall (u,v) \in E, \forall m \in M, &&
\end{aligned}$$

where the objective value is denoted by σ and the values b^m_u are constructed to satisfy the node balance constraints and are defined as:

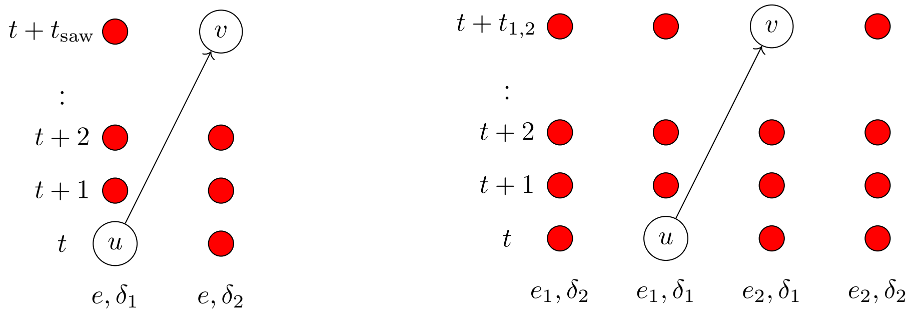
$$b^m_u = \begin{cases} 1, & \text{if } u = s_m \\ -1, & \text{if } u = t_m \\ 0, & \text{otherwise.} \end{cases}$$

In the ILP, '(4.4) - (4.8)' refers to the additional constraints which are introduced subsequently. Constraints (4.4), (4.5) and (4.6) are essential for modeling the Shunt Routing Problem properly. They are all of the *knapsack* type, where at most one of the decision variables in the constraints is allowed to have value 1. Constraints (4.7) and (4.8) are non-fundamental constraints which are introduced to reduce the solution space and speed up the computation. The objective function minimizes over the weights of the edges chosen in the routes. By choosing the weights for unplanned movements ($\omega_m^{\text{unplanned}}$) sufficiently large, the primary goal is to minimize the number of unplanned movements.

- **A/B directions**: An infra element e at time instant t cannot be used in both directions A and B simultaneously. To model this, we restrict that there is at most one edge used pointing out of both vertices, $v_{e,t,A}$ and $v_{e,t,B}$. For any infra element e which can be approached in both directions (i.e. $D_e = \{A, B\}$) and for all considered time instants t, add the following constraint:

$$\sum_{m\in M} \sum_{v\in \text{OUT}(v_{e,t,A})} x^m_{v_{e,t,A},v} + \sum_{m\in M} \sum_{v\in \text{OUT}(v_{e,t,B})} x^m_{v_{e,t,B},v} \leq 1. \qquad (4.4)$$

Here $\text{OUT}(u) \subseteq V$ denotes the set containing all nodes to which an outgoing edge of $u \in V$ is pointing (i.e. $\text{OUT}(u) = \{v \in V \ : \ (u, v) \in E\}$).



(a) $(u, v) \in E_{\text{saw}}$

(b) $(u, v) \in E_{\text{move}}$

Figure 4.8: Illustration of Constraint (4.6): when using $(u, v) \in E_{\text{saw}} \cup E_{\text{move}}$, the vertices marked in red are not available for other movements.

- **Crossings**: Two crossing tracks such as tracks 7 and 8 in Figure 4.4 cannot be used simultaneously. The construction of the constraint to avoid that crossing tracks e_1, e_2, are used simultaneously at time instants t, is similar to (4.4). This constraint ensures that there is at most one edge used in the set of outgoing edges of the nodes corresponding both tracks and both directions, $v_{e_1,t,A}, v_{e_1,t,B}, v_{e_2,t,A}$ and $v_{e_2,t,B}$:

$$\sum_{m \in M} \sum_{v \in \text{OUT}(v_{e_1,t,A})} x^m_{v_{e_1,t,A},v} + \sum_{m \in M} \sum_{v \in \text{OUT}(v_{e_1,t,B})} x^m_{v_{e_1,t,B},v} + \sum_{m \in M} \sum_{v \in \text{OUT}(v_{e_2,t,A})} x^m_{v_{e_2,t,A},v} + \sum_{m \in M} \sum_{v \in \text{OUT}(v_{e_2,t,B})} x^m_{v_{e_2,t,B},v} \leq 1. \tag{4.5}$$

- **Blocking during movements**: When an edge from element e_1 at time t to element e_2 at time $t+T$ is used in a path, nodes corresponding elements e_1 and e_2 in the time interval $[t, t+T]$ are not available for other paths. We mark these intermediate nodes as *forbidden*. Figure 4.8 illustrates this idea by depicting *forbidden* nodes in red. Sawing movements and movements between different infra elements are distinguished here.

  For any edge $(u, v) \in E_{\text{saw}}$, with $u = v_{e,t,\delta_1}$ and $v = v_{e,t+t_{\text{saw}},\delta_2}$ $(\delta_1 \neq \delta_2)$, the set of *forbidden* nodes can be denoted by $\bar{V}_{(u,v)} = \{v_{e,t+\bar{t},\delta_1} \mid 1 \leq \bar{t} \leq t_{\text{saw}}\} \cup \{v_{e,t+\bar{t},\delta_2} \mid 0 \leq \bar{t} \leq t_{\text{saw}} - 1\}$.

  For all edges $(u, v) \in E_{\text{move}}$, with $u = v_{e_1,t,\delta_1}$ and $v = v_{e_2,t+t_{1,2},\delta_1}$, the set of *forbidden* nodes can be denoted by $\bar{V}_{(u,v)} = \{v_{e_1,t+\bar{t},\delta_1} \mid 1 \leq \bar{t} \leq t_{1,2}\} \cup \{v_{e_1,t+\bar{t},\delta_2} \mid 0 \leq \bar{t} \leq t_{1,2}\} \cup \{v_{e_2,t+\bar{t},\delta_1} \mid 0 \leq \bar{t} \leq t_{1,2} - 1\} \cup \{v_{e_2,t+\bar{t},\delta_2} \mid 0 \leq \bar{t} \leq t_{1,2}\}$.

  For all $(u, v) \in E_{\text{saw}} \cup E_{\text{move}}$, for all forbidden nodes $w \in \bar{V}_{(u,v)}$, the following constraint is added to ensure that all edges pointing out $w \in \bar{V}_{(u,v)}$ are idle, when $(u, v) \in E_{\text{move}}$ is used in the solution:

$$\sum_{m \in M} x^m_{u,v} + \sum_{m \in M} \sum_{h \in \text{OUT}(w)} x^m_{w,h} \leq 1 \tag{4.6}$$

  Note that Constraint (4.6) makes that Constraint (4.2) is redundant for $(u, v) \in E_{\text{saw}} \cup E_{\text{move}}$.

- **Parking tracks for parking trains only**: We enforce that entering a parking track is restricted to trains which are assigned to this parking track. Therefore we set all decision variables corresponding forbidden movements to zero. So for all edges (i, j) which correspond to a movement at a parking track where $m \in M$ is not assigned to, we set:

$$x^m_{i,j} = 0 \tag{4.7}$$

- **Arrival at parking tracks**: We want to enforce that once a train enters its parking track, it is forced to drive immediately (i.e. without using waiting edges) to its assigned position (k) at the parking track. Figure 4.9 illustrates this idea: the train arriving at τ_{park} at time $t = 1$ is not allowed to follow the path marked in red. Say that an edge $(i_0, i_1) \in E$ is called *entering edge* if and only if it corresponds to a movement from non-parking to parking track. Then for every entering edge (i_0, i_1) with $i_0 = v_{e,t,s}$ and $i_1 = v_{\tau_{\text{park},1},t+T,s}$ and movement m assigned to τ_{park}, holds that if $x^m_{i_0,i_1} = 1$, then $x^m_{i_1,i_2} = \ldots = x^m_{i_{(k-1)},i_k} = 1$, where $i_j = v_{\tau_{\text{park},j},t+T,s}$ $(j = 1, \ldots, k)$. In words this means that if movement m enters the first position at its parking track τ_{park}, the constraints forces the movement to drive to the assigned k^{th} position immediately (i.e., in the same time instant). This leads for every movement $m = (\tau_{\text{platform}}, \tau_{\text{park}}, k, r, d)$, for every entering edge (i_0, i_1) with $i_1 = v_{\tau_{\text{park},1},t+T,s}$, to the constraints:

$$\begin{aligned} x^m_{i_0,i_1} &\leq x^m_{i_1,i_2} \\ x^m_{i_0,i_1} &\leq x^m_{i_2,i_3} \\ &\vdots \\ x^m_{i_0,i_1} &\leq x^m_{i_{(k-1)},i_k}. \end{aligned} \tag{4.8}$$

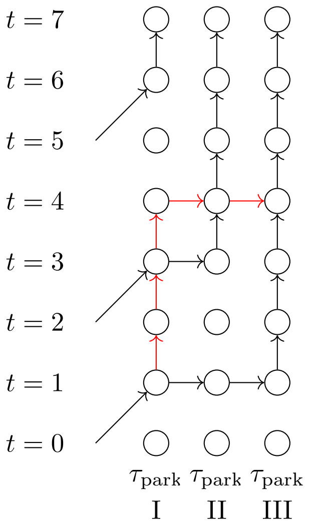


Figure 4.9: Constraint (4.8): Arrival at parking tracks: once entering the parking track, trains are forced to drive to their position immediately.

4.3 Computational Complexity

In this section we study the computational complexity of our problem. As mentioned at the beginning of this chapter, Karp (1975) proved the NP-completeness of the DPP. What about the complexity of the additional constraints? Note that the essential additional constraints (Constraint (4.4) for A/B directions, Constraint (4.5) for crossings and Constraint (4.6) for blocking during movements) can all be formulated as forbidden pairs constraints, where for a given set of pairs at most one node per pair can be used in the solution. So our problem can be interpreted as an extended version of the NP-complete Impossible Pair constrained Path problem (IPP):

Impossible Pair constrained Path problem (IPP)
Instance: A directed graph $G = (V, E)$, a source $s \in V$, a sink $t \in V$ and a set of forbidden pairs $F = \{\{a_i, b_i\} \in V \times V |\ 1 \leq i \leq n\}$.
Question: Does there exist a path from s to t in G, such that for every pair $\{a_i, b_i\} \in F$ at most one node is used?

Gabow (1976) proves that the IPP is NP-complete. In this section, we briefly recap the main ideas of the NP-completeness proofs for the DPP and the IPP.

Idea of the proof NP-completeness DPP

Karp (1975) constructed his proof about the NP-completeness of the DPP by a reduction from the 3-SAT problem. Given a Boolean expression $B = (p_{11} \vee p_{12} \vee p_{13}) \wedge (p_{21} \vee p_{22} \vee p_{23}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee p_{m3})$, the graph $G = (V, E)$ as input of the DPP is obtained by joining together a number of subgraphs, each corresponding to a variable of B. Let x be a variable in B which occurs in clauses $i_1, i_2, \ldots, i_p$ and suppose $\bar{x}$ occurs in clauses $j_1, j_2, \ldots, j_q$, then the subgraph corresponding x is constructed as shown in Figure 4.10.

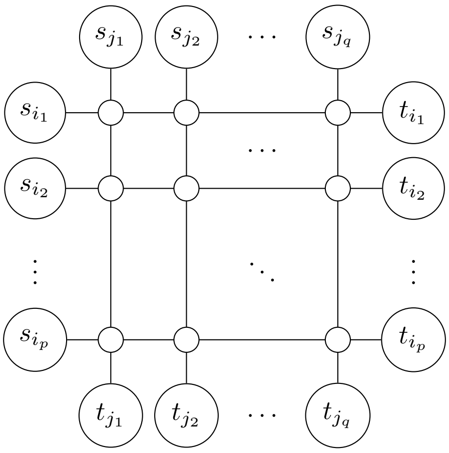


Figure 4.10: Subgraph for variable x.

Source-sink pairs are in one-to-one correspondence with the clauses in the given satisfiability problem: pair (s_i, t_i) corresponds to clause C_i. Using a horizontal path from s_{i_l} to corresponding sink t_{i_l} in Figure 4.10 corresponds to setting x = true. Similarly, using a vertical path from s_{j_k} to t_{j_k} corresponds to setting x = false. However, using one horizontal path excludes the option of using a vertical path and vice versa (i.e. x cannot be set true and false simultaneously).
The overall graph G is obtained by simply identifying, as a single node, all the occurences of each s_i (or t_i) in the subgraphs for all variables. Every clause $C_i = (p_{i1} \vee p_{i2} \vee p_{i3})$ is true if and only if at least

one of the three literals is set true. This is similar to: there is a node disjoint s_i,t_i-path in G if and only if there is a node disjoint s_i,t_i-path in at least one of the subgraphs constructed from the variables of p_{i1}, p_{i2} or p_{i3}. It can be proved that satisfiability of B implies that there exist a feasible solution for the DPP and vice versa. Note that the construction can be done in polynomial time. Therefore we can conclude the NP-hardness of the DPP.

Idea of the proof NP-completeness IPP

Gabow (1976) proves that the IPP is NP-complete by a reduction from the 3-SAT problem. He shows that the problem remains NP-hard even when the underlying graph G is acyclic (which is the case for the time-expanded graph in our model) and all in- and out-degrees are at most two. We briefly reproduce the main idea of the proof.

Given a Boolean expression B in 3-conjunctive normal form $B = (p_{11} \vee p_{12} \vee p_{13}) \wedge (p_{21} \vee p_{22} \vee p_{23}) \wedge \ldots \wedge (p_{m1} \vee p_{m2} \vee p_{m3})$, we construct a graph $G = (V, E)$ as input of the IPP. V consists of a source s, a sink t and a node v_{ij} for every literal p_{ij} in B. So formally

$$V = \{s, t\} \cup \{v_{ij} |\ 1 \leq i \leq m,\ 1 \leq j \leq 3\}.$$

E contains all pairs of literals in consecutive clauses of B, considering s as the first clause of B and t as the last. Thus

$$\begin{aligned} E = &\{(s, v_{1j})|\ 1 \leq j \leq 3\} \cup \\ &\{(v_{ij}, v_{i+1,k})|\ 1 \leq i \leq m,\ 1 \leq j,\ k \leq 3\} \cup \\ &\{(v_{mj}, t)|\ 1 \leq j \leq 3\}. \end{aligned}$$

The set of forbidden pairs, F, contains all pairs of literals that are negations of each other:

$$F = \{\{v_{ij}, v_{kl}\}|\ p_{ij} = \bar{p_{kl}}\}.$$

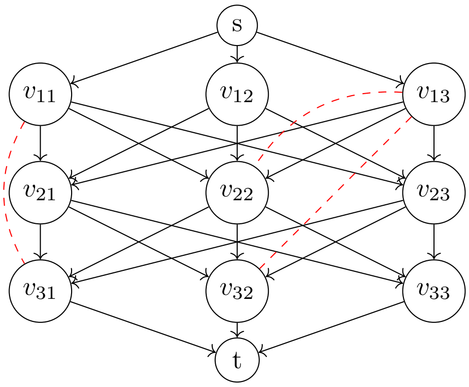


Figure 4.11: Graph G constructed from the Boolean expression $B = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee d) \wedge (\bar{a} \vee \bar{c} \vee e)$. Pairs in F are connected via a dotted red arc.

Figure 4.11 illustrates the construction for $B = (a \vee b \vee c) \wedge (b \vee \bar{c} \vee d \wedge (\bar{a} \vee \bar{c} \vee e)$. For this example, $F = \{\{v_{11}, v_{31}\}, \{v_{13}, v_{22}\}, \{v_{13}, v_{32}\}\}$. Now consider a constrained path P from s to t. The path goes through a *stage* for every clause of B. By making the literals corresponding the nodes in P all true in B, the value of B is true. Because P satisfies the constraints F, these assignments are non-conflicting. Hence the existence of P implies satisfiability of B. Similarly, it can be shown that the satisfiability of B implies the existence of an impossible pair constrained path. Note that G and F can be constructed in polynomial time from B. From the NP-completeness of 3-SAT can then be concluded that IPP is NP-hard. To show that is NP-complete, we need to show that IPP is in NP, which is easy to check.

4.4 Time discretization

To transform the topology network to a time-expanded network $G = (V, E)$ as described in Section 4.1, we need to discretize time. In $G = (V, E)$, nodes are duplicated for every time step for the chosen granularity. In the most straightforward version, the granularity is chosen equivalently for every infra element during the whole considered time horizon $[T_0, T_f]$. This leads to a lower bound for $|V|$: $(\lfloor \frac{T_f - T_0}{k} \rfloor \times |\text{infra elements}|)$, where k denotes the fixed size of the time steps in the chosen granularity. An upper bound for $|V|$ is $((\lceil \frac{T_f - T_0}{k} \rceil \times |\text{infra elements}| \times 2) + |M|)$. This upper bound is based on the situation that all infra elements can be approached from two directions and for all movements $m \in M$ a source node is created. To create more advanced time grids, it is possible to vary the time discretization over different time intervals or infra elements.

Choosing a proper granularity is balancing between computation time and quality of the solution. The sparser the underlying time grid, the smaller the ILP, the smaller the time-expanded network, the shorter the computation time to find a solution. However, sparse granularity might decrease the quality of the solution. To illustrate this, Figure 4.12 depicts an example of a movement between two infra elements, e_1 and e_2. This movement has a duration of 40 seconds. The chosen time grid creates for both e_1 and e_2 a node for every 30 seconds ($k = 30$), starting at time 00:00. The dashed lines show the movements without delay. However no node represents the actual arrival time and thus these edges cannot be added to the network. This can be solved by rounding the arrival time to the nearest time instant in the chosen granularity. The resulting movement can be interpreted as a delayed movement. The colored fields in Figure 4.12 depict the delay caused by these rounding errors. Figure 4.13 shows an example of a movement via five infra elements. This example illustrates that rounding errors might cause a lot of delay, especially when the granularity is chosen sparse.

In theory, rounding errors can be completely avoided by choosing k equal to the greatest common divisor of the travel times between all adjacent infra elements in the topology.

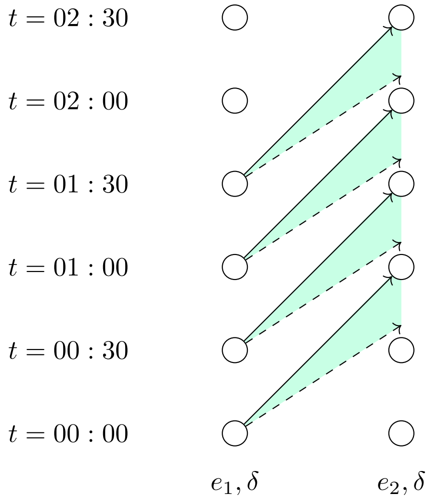


Figure 4.12: Example of a delayed movement from e_1 to e_2 (direction δ). Actual time to travel from e_1 to e_2: 40 seconds. Colored fields illustrate delay.

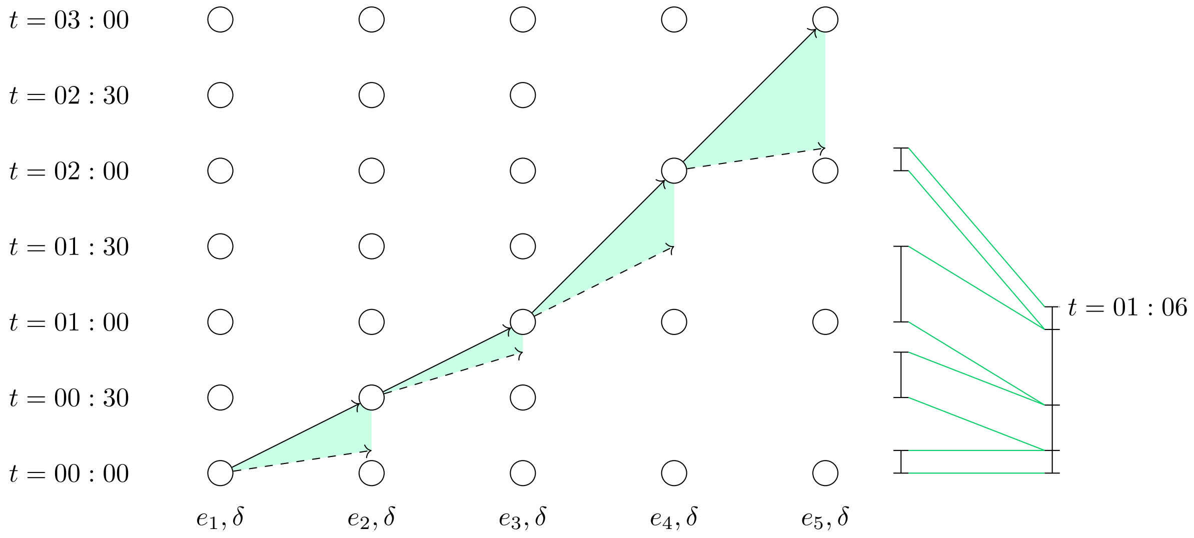


Figure 4.13: Example of a delayed movement. Colored fields illustrate delay. Without delay the train could arrive its destination at $t = 01:06$ instead of $t = 03:00$.

Chapter 5

Computational results with the Disjoint Shortest Paths algorithm

This chapter presents the experiments performed on the Disjoint Shortest Paths algorithm (DSP). We compare the results of DSP with the results of two other methods: the greedy Successive Shortest Paths algorithm (SSP) and an algorithm based on the current implementation in OPG+, RouteZoeker (RZ). The benchmark is introduced in Sections 5.1 and 5.2, where respectively the used data sets and the solution methods SSP and RZ are introduced. Computational results are presented and discussed in Section 5.3.

5.1 Description of the benchmark: data sets

We perform experiments on two locations: station Enschede, which is a midsize station in the Netherlands, and station Eindhoven which is one of the biggest Dutch stations. In Table 5.1 some characteristics of both stations are shown. We distinguish three types of infra elements: switches, tracks and track parts. This last category consists of parts of the track which are not assigned a *name* or number. The layout of both stations is shown in Figure 5.1. Enschede consists of 51 infra elements in total, Eindhoven contains 295 infra elements. In the Appendix the layout of both locations are shown in detail (see Figures 8.1 and 8.2).

To benchmark our algorithm, we created a set of twelve instances to the Shunt Routing Problem of which six are on station Enschede and six on station Eindhoven. The characteristics of these twelve instances are shown in Table 5.2. Data sets A, B, G and H are relatively *small* instances, in terms of $|M|$. Instances E and L are instances based on large sets of real data. To challenge our algorithms, we transformed these sets to respectively data sets F and K, by shifting the time windows of the movements to a much denser scheme. The set of movements C, D, I and J are artificially created, again to challenge the capacities of the algorithms. They contain an -unrealistically- dense set of movements within a relatively short time interval. The details of the instances can be found in the Appendix in Tables 8.3 - 8.14.

Table 5.1: Characterizations of the considered stations

Location	**#tracks**	**#switches**	**#track parts**	**#infra elements**
Enschede	13	18	20	51
Eindhoven	66	107	122	295

We list settings used in the performed experiments.

- We assume there are no reservations (i.e. $R = \emptyset$).
- The considered time horizon is deduced from the first release and the latest deadline of the movements in M. We discretize time for every infra element in the considered time horizon with steps of 10 seconds.
- The order at the park track ('k' in the description of a movement $m \in M$) is determined from the order of release. We assume that the trains arrive at their park track in order of release.
- We set $\omega_m^{\text{unplanned}} = 10,000,000$ for all movements $m \in M$. When no route can be found, a movement m is forced to take the *expensive* dummy edge with this weight. This is weight is of such a high order compared to other weights, that the primary objective becomes minimizing the number of unplanned movements. Weights $\omega_\tau^{\text{wait}}$ and ω_τ^{saw} for waiting and sawing a track τ can be found in Tables 8.1 and 8.2 in the Appendix.
- We set the time needed for a sawing movement at 2 minutes, traverse times of switches are 0 seconds and traverse times of track parts 10 seconds. The traverse times (in terms of seconds) of tracks are determined in the following way:

$$\text{traverse time}(\tau) = 10 \cdot \lceil \frac{\text{length}(\tau)}{100} \rceil.$$

  So for example, a track with length 451 meter has a traverse time of 50 seconds. Lengths and traverse times of the tracks can be found in Tables 8.1 and 8.2 in the Appendix.
- We restrict waiting to tracks only (i.e. it is not allowed to wait at a switch or track part).

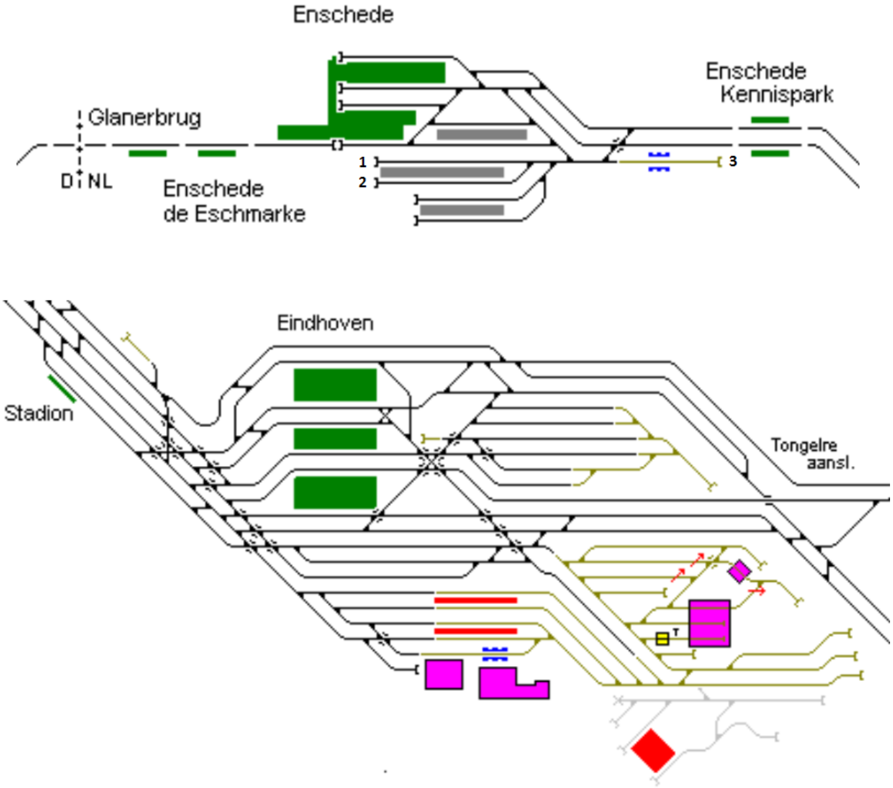


Figure 5.1: Infrastructure of station Enschede and station Eindhoven.

Table 5.2: Characteristics of the considered instances. $|M|$ denotes the number of movements to be planned, $\overline{d_m - r_m}$ denotes the average length of the time windows of the movements ([hh:mm:ss]), $|V|$ and $|E|$ denote the number of nodes respectively edges in the underlying time-expanded graph.

Set	**Location**	$\|M\|$	$\|V\|$	$\|E\|$	**Time horizon**	$\overline{d_m - r_m}$
A	Enschede	13	13578	18790	20h00 - 20h20	00:08:18
B	Enschede	12	13455	18782	20h00 - 20h20	00:07:20
C	Enschede	16	20666	29633	20h00 - 20h30	00:22:30
D	Enschede	32	23594	35377	20h00 - 20h30	00:23:00
E	Enschede	27	542988	805529	18h19 - 06h17	01:45:42
F	Enschede	21	198162	288687	18h19 - 22h54	00:56:03
G	Eindhoven	14	111524	160995	20h00 - 20h30	00:15:00
H	Eindhoven	10	472525	688327	18h05 - 20h15	00:44:48
I	Eindhoven	28	112638	165051	20h00 - 20h30	00:22:30
J	Eindhoven	28	112588	164951	20h00 - 20h30	00:21:00
K	Eindhoven	38	236931	349616	18h16 - 19h18	00:14:08
L	Eindhoven	45	1803468	2675598	18h16 - 02h04	00:15:01

5.2 Description of the benchmark: Successive Shortest Paths and RouteZoeker algorithm

We compare the Disjoint Path approach with two different methods: the Successive Shortest Paths (SSP) algorithm which greedily schedules the movements after sorting them chronologically and the RouteZoeker (RZ) algorithm which is based on and named after the submodule which solves the Shunt Routing Problem in the NS-tool OPG+ (see Figure 1.4). Both algorithms are implemented such that the structure of the time-expanded graph $G = (V, E)$ and the same parameters are used.

Figure 5.2 shows the global layout of the SSP algorithm. Shortest (s_m, t_m)-paths in G are found using Dynamic Programming (using the fact that G is acyclic). Note that it does not suffice to remove the nodes and edges in the found path. Beside these nodes and edges, also nodes and edges corresponding the three fundamental additional constraints (Constraints 4.4, 4.5 and 4.6) need to be removed: corresponding A/B directions, crossings and blocking during movements. Note that there always exist an

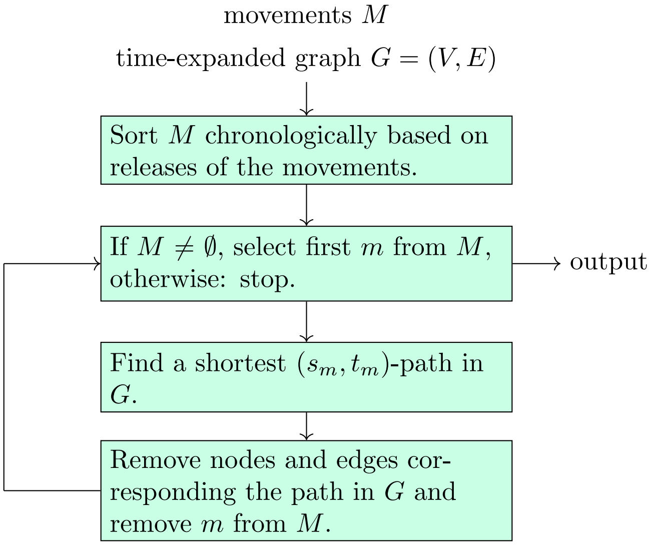

movements M
time-expanded graph $G = (V, E)$
Sort M chronologically based on releases of the movements.
If $M \neq \emptyset$, select first m from M, otherwise: stop.
output
Find a shortest (s_m, t_m)-path in G.
Remove nodes and edges corresponding the path in G and remove m from M.

Figure 5.2: Global layout of Successive Shortest Paths (SSP) algorithm.

s_m, t_m-path in G, since we added dummy arcs to G between every (s_m, t_m)-pair.

The layout of the RZ algorithm is shown in Figure 5.3. The main idea of this algorithm is that the route in terms of location is fixed beforehand and that planning times are determined integrally. In the first step, for every $(\tau_{\text{platform}}, \tau_{\text{park}})$-pair a shortest path in the topology network is found using Dynamic Programming. In OPG+ the shortest routes are determined in submodule GSSCG (see Figure 1.4). For platform tracks which can be departed from both sides, two routes are fixed. For other platform tracks, there is one fixed route. Using an ILP similar to the ILP for the DSP, for every movement $m \in M$ a planning time is determined for (one of) the fixed route(s). This slightly differs from the actual implementation of RouteZoeker in OPG+, where this problem is solved in two MIPs sequentially (first MIP for determining the leaving side of the platform, second MIP for finding a planning time).

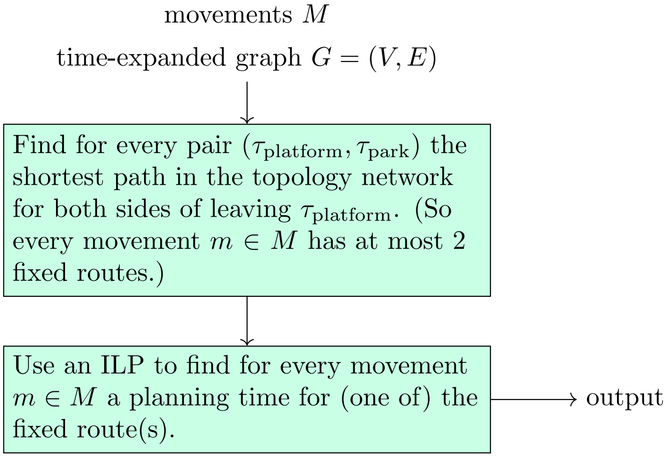

movements M
time-expanded graph $G = (V, E)$
Find for every pair $(\tau_{\text{platform}}, \tau_{\text{park}})$ the shortest path in the topology network for both sides of leaving τ_{platform}. (So every movement $m \in M$ has at most 2 fixed routes.)
Use an ILP to find for every movement $m \in M$ a planning time for (one of) the fixed route(s).
output

Figure 5.3: Global layout of RouteZoeker (RZ) algorithm.

5.3 Computational results

For all computations in this report, we used CPLEX version 12.6.2 on a computer with processor Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz (6 CPUs), 3.6GHz and memory 16384MB RAM. We present the major findings of the computational results in this and the next chapters colored boxes.

We performed experiments on the twelve instances for the Disjoint Shortest Paths, the Successive Shortest Paths and the RouteZoeker algorithm (respectively abbreviated by DSP, SSP and RZ). Table 5.3 gives an overview of the computational results. Results for DSP and RZ of instance L are missing, because the memory needed for the ILP exceeded the limits of CPLEX. The runs on D, I and J for DSP are stopped after 10 hours of computation time. Results in the table for these experiments are based on the best solution found within this computation period and the best solution found within the first 15 minutes of computation. In the table, σ denotes the objective value of the ILP solution. This value is the sum of the weights of the used edges from the time-expanded network (see Section 4.2). σ is mainly determined by the number of dummy edges (i.e. unplanned movements) in the solution, and partly by sawing and waiting costs.

Statement 1: In comparison to the benchmark, DSP succeeds in finding significantly better routing schedules, where less movements are left unplanned. However, for large instances the algorithm needs long computation times.

The last row of Table 5.3 summarizes the results by presenting the cumulative results of $|M_{\text{unp}}|$ and the CPU time. (Experiment L is not included, for experiment D, I and J the solution after 15 minutes computation time is included.) DSP succeeds in finding schedules for which more movements are found an execution time and route: $|M_{\text{unp}}|$ for DSP is 57 compared to respectively 136 and 84 for SSP and RZ. These values show the potential of this new approach. However, we also see a difference in required computation time needed.

The results of DSP and SSP in Table 5.3 are marked green if the result is strictly better than the result of RZ and red if the result is strictly worse than the result of RZ. In all experiments except L, SSP is outperformed by DSP and in most experiments by RZ as well. Only in experiments B and G, SSP is outperforming RZ. We observe the promising result that $|M^{\text{DSP}}_{\text{unp}}| < |M^{\text{RZ}}_{\text{unp}}|$ for experiments A, B, C and G. For experiment K we observe that, although $|M^{\text{DSP}}_{\text{unp}}| = |M^{\text{RZ}}_{\text{unp}}|$, the objective value is improved, i.e. $\sigma^{\text{DSP}} < \sigma^{\text{RZ}}$. So the quality of the routes found by DSP is (slightly) better than the routes found by RZ.

For experiments E, F and H, both RZ and DSP find a schedule where all movements can be executed, objective values are equal. For experiments D, I, J which are stopped before finishing, we see that the solution found after 15 minutes for D and J is even better than the solution found by RZ. This is not the case for experiment I, where RZ is only outperformed by DSP after a long computation time.

We zoom in to the results of instances C and D which are artificially created, dense sets of movements (see Table 8.5 and 8.6 in the Appendix). In experiment C, 16 movements need to be scheduled within the short period from 20h00 to 20h30. SSP, RZ and DSP find a schedule where respectively 4, 8 and all of the movements are executed. This experiment shows that extra freedom of the DSP can double the number of planned movements in the schedule compared to RZ. Fixing the routes beforehand limits for example the choice of the saw track, which might be the bottleneck in this situation. However, the computation time of DSP is almost 100 minutes, which is too long for practical implementation.
Experiment D challenges the algorithms even more, by doubling the number of movements to be scheduled to 32. Where SSP and RZ already reached their limitations in experiment C and do not succeed in scheduling more movements, DSP finds a solution where 19 movements can be executed. As we expected, the computation time of the algorithm is extremely increased.

Table 5.3: Computational results of the benchmark on DSP, SSP and RZ. The objective value is denoted by σ and $|M_{\text{unp}}|$ denotes the number of movements for which no route is found. Results of DSP and SSP are marked green if the result is strictly better than the result of RZ and red if the result is strictly worse than the result of RZ. [1]: experiment is stopped after 10 hours of computation time. *: total of sets (excluding result L), for set D, I and J the results after 15 minutes are included.

		DSP			**SSP**			**RZ**		
Set	$\lvert M \rvert$	σ	$\lvert M_{\textbf{unp}} \rvert$	**CPU time** [hh:mm:ss]	σ	$\lvert M_{\textbf{unp}} \rvert$	**CPU time** [hh:mm:ss]	σ	$\lvert M_{\textbf{unp}} \rvert$	**CPU time** [hh:mm:ss]
A	13	390054	0	00:00:17	60180027	6	00:00:01	60090028	6	00:00:08
B	12	20380000	2	00:00:04	50230000	5	00:00:03	70090000	7	00:00:07
C	16	480000	0	01:39:18	120120000	12	00:00:05	80080000	8	00:01:37
D[1]	32	160120936	16	00:15:00	280120000	28	00:00:06	240080000	24	00:05:42
		130171132	13	10:00:00						
E	27	260563	0	01:06:56	130130000	13	00:03:20	260563	0	00:45:19
F	21	200567	0	00:10:08	120080000	12	00:01:12	200567	0	00:08:09
G	14	50420	0	00:01:49	50520	0	00:00:25	20030198	2	00:01:34
H	10	40000	0	00:07:43	20033260	2	00:00:07	40000	0	00:08:58
I[1]	28	170110852	17	00:15:00	210041390	21	00:00:46	150131672	15	00:05:47
		150131098	15	10:00:00						
J[1]	28	160120936	16	00:15:00	220061940	22	00:00:47	160121309	16	00:03:58
		130151766	13	10:00:00						
K	38	60170160	6	01:33:25	150050550	15	00:02:59	60171510	6	00:09:36
L	45	*error*			50120000	5	00:23:43	*error*		
Total*			**57**	**05:24:40**		**136**	**00:09:51**		**84**	**01:30:55**

Chapter 6

Heuristic methods to speed up the algorithm

In Chapter 5, the computational results of experiments on the Disjoint Shortest Paths algorithm have been presented. For implementation, it is desirable to have smaller computation times. To decrease the CPU time, we investigate several heuristic methods. In Section 6.1 we present a simple *shifting deadlines* heuristic which introduces an upper bound for the length of a time window $[r_m, d_m]$ in which a movement needs to be planned. In Section 6.2 multiple *decomposition algorithms* are presented which decompose the input M into multiple smaller instances which are solved subsequently. Finally, a *rolling horizon heuristic* is presented in Section 6.3.

The computational results presented in this chapter are found under the same conditions and for the same instances as presented in Section 5.1.

6.1 Shifting deadlines

Observing data set E on station Enschede based on real data, we notice that some movements have a large time window $[r, d]$ in which the movement can be executed. Six of the 27 movements of instance E have time window with length of around five hours. The average length of the time intervals is 1h45. To speed up the algorithm, we experiment with shifting deadlines to reduce the number of decision variables $x^m_{u,v}$. We introduce a parameter, $t_{\max}$, which determines the maximum length of a time window. So for any movement $m \in M$ with time window $[r_m, d_m]$, whenever $d_m > r_m + t_{\max}$, we shift the deadline to $d_m := r_m + t_{\max}$.

We performed experiments on instances E, F and H for different values of $t_{\max}$ to investigate the quality of the solution and the computation time of the algorithm. Instance F contains 21 movements with an average length of the time intervals, $\overline{d_m - r_m}$, of 56 minutes. Instance H is an artificially created instance on location Eindhoven, containing 10 movements with $\overline{d_m - r_m} \approx 45$ minutes.

Statement 2: Eliminating decision variables by the *shifting deadlines* heuristic drastically reduces the computation time of the algorithm, hardly losing quality of the solution.

The results of the experiments are shown in Table 6.1. The smallest value of $t_{\max}$ for which a solution is found with $|M_{\text{unp}}| = 0$ is shown in green. For all instances, we observe a significant improvement in terms of computation time. Where the original DPP algorithm needs more than an hour to find a solution for experiment E, the experiments with shifted deadlines are finished within 7 minutes. When $t_{\max}$ is chosen too small, the DPP algorithm does not succeed in finding a route for all movements. However for $t_{\max} \geq 14$ minutes, the algorithm finds a schedule where all movements can be executed.

For instance F, we found that for $t_{\max} \geq 12$ minutes, the algorithm finds a schedule where all movements can be executed. For $t_{\max} = 12$, the computation time is reduced with a factor approximately 9. For instance H, this is the case for $t_{\max} = 6$ minutes, where the computation time is reduced with a factor approximately 7.

Table 6.1: Computational results of the DPP algorithm with shifting deadlines heuristic.

Set			$t_{\max}$	σ	$\lvert M_{\text{unp}}\rvert$	**CPU time**
	$\lvert M\rvert$	$\overline{d_m - r_m}$				[hh:mm:ss]
E	27	01:45:42	no shifting	260563	0	01:06:56
			12 min	20280251	2	00:04:57
			13 min	10290251	1	00:04:59
			14 min	300251	0	00:05:03
			15 min	260563	0	00:05:31
F	21	00:56:03	no shifting	200567	0	00:10:08
			10 min	20220000	2	00:00:58
			11 min	10230000	1	00:01:03
			12 min	240000	0	00:01:08
			15 min	200567	0	00:01:16
H	10	00:44:48	no shifting	40000	0	00:07:43
			4 min	40000000	4	00:00:56
			5 min	40000000	4	00:01:06
			6 min	40000	0	00:01:08
			10 min	40000	0	00:01:35

6.2 Decomposition algorithms

Another method to speed up the algorithm is to decompose the instance M in multiple smaller instances and solve the problem for these instances subsequently. The main layout of this decomposition heuristic is depicted in Figure 6.2. In every loop, we select a subset $M^{\text{sub}} \subseteq M$ and perform the DPP algorithm on this subset of movements. We remove M^{sub} from M and transform the found routes into reservations in R. Then we return to selecting a new subset $M^{\text{sub}} \subseteq M$. We stop the algorithm when M is empty. Selecting $M^{\text{sub}} \subseteq M$ can be done in various ways. Figure 6.1 depicts the global idea of the selection procedure we used. After sorting M, we select a subset of M. We investigated sorting based on release times, deadlines and size of the time horizons of the movements. In Table 6.2 the mathematical description of the used sorting methods is denoted. The selection methods are shown in Table 6.3. For the selection, we introduce two parameters: t_Δ and k.

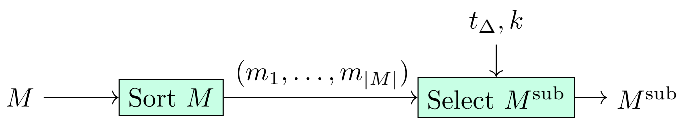


Figure 6.1: Selecting $M^{\text{sub}} \subseteq M$

Table 6.2: Methods for sorting M to $(m_1, m_2, \dots, m_{|M|})$. In the description, i and j denote indices assigned to movements.

Sorting method	**Mathematical description**	
Release, forward	$i < j \leftrightarrow r_{m_i} < r_{m_j}$	for $0 \leq i, j \leq \lvert M \rvert$
Release, backward	$i < j \leftrightarrow r_{m_i} > r_{m_j}$	for $0 \leq i, j \leq \lvert M \rvert$
Deadline, forward	$i < j \leftrightarrow d_{m_i} < d_{m_j}$	for $0 \leq i, j \leq \lvert M \rvert$
Deadline, backward	$i < j \leftrightarrow d_{m_i} > d_{m_j}$	for $0 \leq i, j \leq \lvert M \rvert$
Size time horizon	$i < j \leftrightarrow (d_{m_i} - r_{m_i}) < (d_{m_j} - r_{m_j})$	for $0 \leq i, j \leq \lvert M \rvert$

Table 6.3: Methods for selecting $M^{\text{sub}} \subseteq M$.

Selection method	**Mathematical description**
Fixed time window$*^1$	$\{m \mid r_m \in [r_1, r_1 + t_\Delta]\ \}$
Fixed size $\lvert M^{\text{sub}} \rvert$ $*^2$	$\{m_i \mid 1 \leq i \leq k\}$
Intersection $*^1$ and $*^2$	$\{m \mid r_m \in [r_1, r_1 + t_\Delta]\ \} \cap \{m_i \mid 1 \leq i \leq k\}$
Union $*^1$ and $*^2$	$\{m \mid r_m \in [r_1, r_1 + t_\Delta]\ \} \cup \{m_i \mid 1 \leq i \leq k\}$
Common park track	$\{m \mid \tau^m_{\text{park}} = \tau^{m_1}_{\text{park}}\}$
Common platform track	$\{m \mid \tau^m_{\text{platform}} = \tau^{m_1}_{\text{platform}}\}$
Smallest time horizon	$\{m_i \mid j - k \leq i \leq j + k$, where m_j has the smallest time horizon $d_{m_j} - r_{m_j}\}$

The methods we investigated in our experiments are summarized in Table 6.4. Decomposition methods 1 - 7 are all based on the idea that M is decomposed into subsets of movements which are *close* to each other in terms of time. For example, method 2 (Table 6.4) first sorts the movements chronologically on their release time, after which it selects the first k movements (for k a fixed parameter). Figure 6.4 illustrates method 2 with an example. The first decomposition method is similar to method 2, except the fact that the size of M^{sub} is not fixed by a parameter, but we select the movements which have a release in a fixed time horizon. Methods 6 and 7 combine the ideas of the first two methods by taking either the intersection or the union of those two subsets.

Methods 8 and 9 select $M^{\text{sub}} \subseteq M$ based on location instead of time. Method 8 (9) selects all movements sharing the park (platform) track of the first movement. The underlying idea of method 10 is to schedule movements which are *difficult* to schedule first. The size of the time horizon in which a movement can be planned, $d_m - r_m$, is used as measure of difficulty. Method 11 (12) combines the idea of method 8 (9) and 10. Method 13 is a variation of method 10. It selects the movement m_j with the smallest time horizon $d_{m_j} - r_{m_j}$ and for a fixed parameter k it determines which $2k$ movements lay around this movement in terms of release time (for the precise definition see Table 6.3).

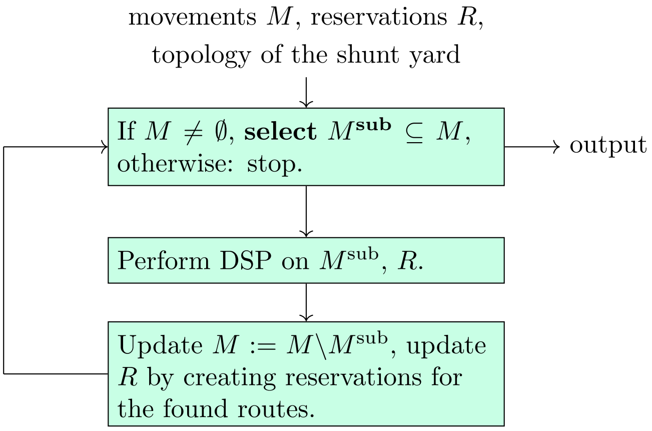

movements M, reservations R,
topology of the shunt yard
If $M \neq \emptyset$, **select M^{sub}** $\subseteq M$, otherwise: stop.
output
Perform DSP on M^{sub}, R.
Update $M := M \backslash M^{\text{sub}}$, update R by creating reservations for the found routes.

Figure 6.2: Global layout of a decomposition heuristic.

Table 6.4: Decomposition methods. Mathematical descriptions can be found in Tables 6.2 and 6.3.

	Sorting	**Selection method**
1	Release, forward	Fixed time window
2	Release, forward	Fixed size $\lvert M^{\text{sub}}\rvert$
3	Release, backward	Fixed size $\lvert M^{\text{sub}}\rvert$
4	Deadline, forward	Fixed size $\lvert M^{\text{sub}}\rvert$
5	Deadline, backward	Fixed size $\lvert M^{\text{sub}}\rvert$
6	Release, forward	Intersection $*^1$ and $*^2$
7	Release, forward	Union $*^1$ and $*^2$
8	-	Common park track
9	-	Common platform track
10	Size time horizon	Fixed size $\lvert M^{\text{sub}}\rvert$
11	Size time horizon	Common park track
12	Size time horizon	Common platform track
13	Release, forward	Smallest time horizon

We limit ourselves to presenting computational results of the most promising methods, which appeared to be methods 3, 10, 11, 12 and 13 (Table 6.4). We performed experiments on the larger instances of our benchmark: C, D, E, F, I, J, K and L. Results are shown in Table 6.5, where green colours indicate the best solutions in terms of the number of unplanned movements, $|M_{\text{unp}}|$, and orange colours indicate the longest computation times.

Statement 3: Decomposition methods in most of the experiments significantly decrease the computation time. For one instance, several decomposition methods found better results within a few minutes, than the DSP (without heuristics) found within 10 hours.

Statement 4: Most decomposition methods avoid the memory problem for large instance L and find solutions within a couple of minutes.

Statement 5: There is no decomposition method which stands out because of consistently performing best compared to other decomposition methods.

In Table 6.5, results of two experiments on L are missing, because the the ILP exceeded the memory capacities of CPLEX. However, most decomposition methods avoid this problem and succeed in finding solutions for instance L within a couple of minutes.

At first glance we see no clear (colour) pattern in the results of Table 6.5. There is no method that stands out because of consistently good or bad performance. For example, method 12 which is based on common platform tracks, performs best compared to other decomposition methods on instances E and F. However, in the other experiments method 12 is outperformed by others. Another example is method 10, which performs much worse than other methods in experiments on E, although it is one of the best methods in experiments on I and J. The performance of the different decomposition methods is probably dependent on the structure of the instance.

Method 3, 10 and 13 are based on parameter k which determines the size of the subsets $M^{\text{sub}} \subseteq M$. Intuitively one would expect that the larger k, the better the quality of the solution. Although this is the case for most of the experiments, increasing the size of the subsets not necessarily improves the quality of the solution. See for example the results of experiment C, method 3. For $k = 4$, a solution is found where all movements can be executed. Increasing k to 5 results in a solution with $|M_{\text{unp}}| = 5$.

Table 6.5: Computational results on different decomposition methods. * : $|M_{\text{unp}}|$. First row shows the results of the DSP without decomposition. For the gray fields, no experiments are performed.

Method	k	C		D		E		F		I		J		K		L	
		*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**
DSP	→	0	01:39:18	13	10:00:00	0	01:06:56	0	00:10:08	15	10:00:00	13	10:00:00	6	01:33:25	*error*	
3	3	6	00:00:15	16	00:00:25	0	00:23:33	8	00:05:10	18	00:03:30	14	00:02:04	8	00:02:44	3	00:07:25
	4	0	00:00:15	14	00:00:20	0	00:21:59	4	00:07:11	17	00:03:12	13	00:01:51	7	00:02:44	2	00:07:58
	5	5	00:00:13	15	00:00:21	0	00:22:35	8	00:05:52	15	00:02:41	13	00:01:43	7	00:03:00	3	00:08:06
	6	2	00:00:16	14	00:00:23					14	00:02:10	13	00:01:58				
	8	0	00:01:26	14	00:18:14					15	00:04:45	13	00:02:04				
10	3	1	00:00:13	14	00:00:32	11	00:18:49	6	00:06:36	15	00:02:13	15	00:02:00	7	00:05:39	3	00:43:03
	4	0	00:00:14	15	00:00:21	11	00:23:43	5	00:07:07	14	00:01:41	14	00:01:57	7	00:04:20	4	00:51:03
	5	2	00:00:13	14	00:00:23	10	00:24:49	5	00:06:41	14	00:02:56	13	00:01:53	6	00:04:51	*error*	
	6	0	00:00:19	13	00:00:59					15	00:09:35	14	00:01:50				
	8	0	00:01:33	13	00:32:33					14	00:06:31	13	00:02:14				
	16	0	00:59:53	12	01:20:59												
11		2	00:00:29	15	00:00:48	9	00:17:59	7	00:03:34	14	00:03:28	14	00:03:16	8	00:05:43	1	00:06:20
12		4	00:00:24	15	00:01:20	0	00:54:20	0	00:09:11	16	00:02:47	16	00:01:47	8	00:05:37	*error*	
13	1	5	00:00:19	17	00:00:29	4	00:14:25	2	00:04:12	18	00:02:39	15	00:02:15	7	00:04:17	2	00:09:51
	2	4	00:00:20	13	00:00:24	5	00:27:08	3	00:06:09	17	00:02:10	14	00:01:57	8	00:04:57	2	00:08:56
	3	0	00:00:20	13	00:02:34	4	00:28:55	4	00:05:36	16	00:02:56	14	00:01:53	7	00:04:08	3	00:10:26
	4	0	00:01:41	13	00:33:45					14	00:16:21	13	00:01:36				
	5	1	00:01:17	14	00:09:31							15	00:01:47				

🟩	best solution	🟧	> 1 hour
🟩	2nd best	🟧	> 30 min
🟩	3rd best	🟧	> 5 min

This might be due to the fact that some movements lay very *close* to each other (in terms of time and location) and therefore it is desirable to schedule these movements simultaneously. When changing k, such a couple of coherent movements can be separated into two different subsets.

For most experiments, we observe a significant drop in the computation time of the decomposition heuristics compared to the time of the DSP without decomposition. For experiments C and D, most computations are finished within a couple of seconds. However, when taking k large, we see that the computation time explodes. Experiments on instances E and F show a relatively small improvement in computation time. Most of the decomposition heuristics on these two instances need over twenty minutes of computation time. This might be due to the fact that for these instances $\overline{d_m - r_m}$ is relatively long. This results in a large number of decision variables. The long computation times of experiments on set L, method 10, might be caused by the fact that the needed memory capacity is approaching the limits of CPLEX.

Experiments with the DSP on instances I, J and D where stopped after 10 hours of computation time (see Chapter 5). Using the decomposition heuristics is not only a way to improve the CPU time of these experiments, for instances I and D there are even results found with lower value $|M_{\text{unp}}|$. Where the DSP without heuristics found for instance I a schedule where 15 movements could not be executed, some of the decomposition methods find a solution where only 14 movements are left unplanned. For experiment D, we see something similar: decomposition method 10 with $k = 16$ finds a solution where one more movements can be executed compared to the results of ten hours running DSP. However, it still takes 80 minutes to find this better solution.

6.3 Rolling horizon approach

We extend the decomposition methods 1 - 7 to rolling horizon algorithms by introducing a new parameter, l, which determines how many additional movements are taken into account when determining the routes. We refer to l as the rolling horizon parameter. The layout of the rolling horizon heuristic is depicted in Figure 6.3. The DPP algorithm is performed on $M^{\text{sub}} \cup M^{\text{RH}}$, but only the routes for movements $m \in M^{\text{sub}}$ are saved. Figure 6.4 shows an example of the rolling horizon method for decomposition method 2 with $k = 3$ and $l = 2$. In the first iteration, the Shunt Routing Problem is solved for movements m_1 upto m_5, of which only the routes for m_1, m_2 and m_3 are saved.

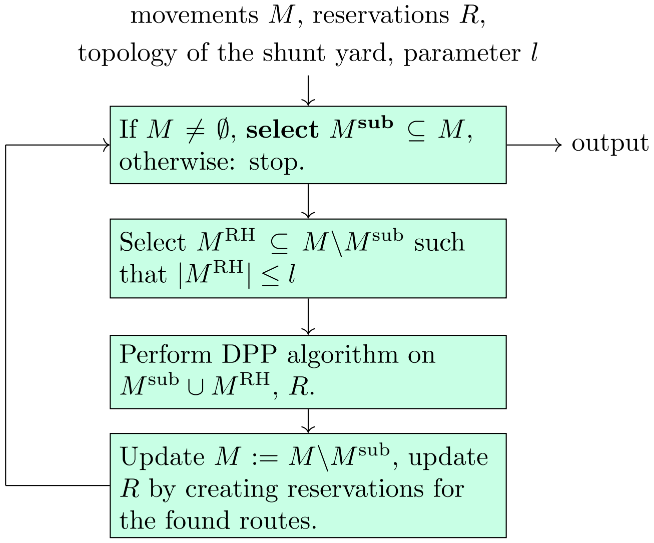

movements M, reservations R,
topology of the shunt yard, parameter l
If $M \neq \emptyset$, **select** $M^{\mathbf{sub}} \subseteq M$, otherwise: stop.
output
Select $M^{\text{RH}} \subseteq M \backslash M^{\text{sub}}$ such that $|M^{\text{RH}}| \leq l$
Perform DPP algorithm on $M^{\text{sub}} \cup M^{\text{RH}}$, R.
Update $M := M \backslash M^{\text{sub}}$, update R by creating reservations for the found routes.

Figure 6.3: Global layout of rolling horizon heuristic.

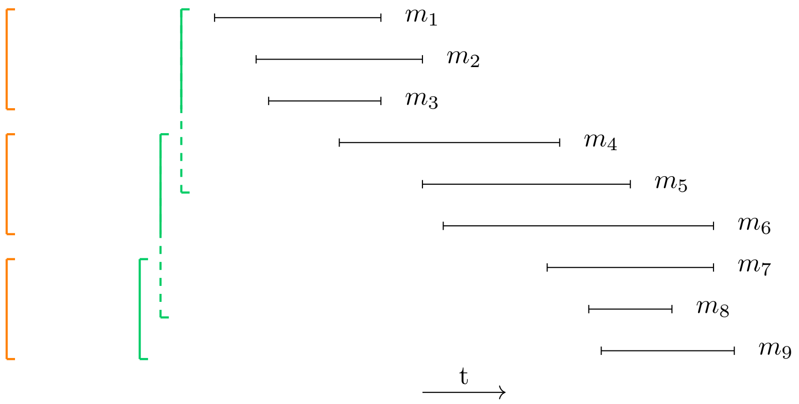


Figure 6.4: Selecting $M^{\text{sub}} \subseteq M$ according to method 2 with fixed parameter $k = 3$. In orange: $l = 0$, in green: $l = 2$.

We present computational results based on decomposition method 3 (Table 6.4), where movements are sorted chronologically (backward) on their release date r_m. As in the previous chapter, we performed experiments on the larger instances of our benchmark: C, D, E, F, I, J, K and L. The results are presented in Table 6.6 with similar colouring as we saw before in Table 6.5.

Statement 6: In most experiments, a larger rolling horizon parameter l results in a higher quality solution, however, increasing l is no guarantee for better solutions.

Overall we see less variation in the results than we saw for the different decomposition algorithms. Intuitively one would probably expect that introducing the rolling horizon parameter l improves the quality of the solution. Although the results confirm this expectation in most cases (see for example experiments on C and F), there are some exceptions. For example, in experiment D the results for $k = 3$ and $k = 5$ are better for $l = 1$ than for $l = 2$. Another example is experiment I, where for $k = 6$ the results of $l = 1$ are better than for $l = 2$. Taking into account movements which lay *close* to the movements in M^{sub}, M^{RH}, might reduce the quality of the routes for movements in M^{sub}. In some cases, this pays off in later iterations, but sometimes it does not.

Roughly we can conclude that the larger the value of l, the longer the computation time. We observe for some experiments a flipping point, where the computation time explodes. This results in computation times of more than (half) an hour. For the experiment of instance D, $k = 8$, $l = 4$, the computation time of more than half an hour results in a solution where $|M_{\text{unp}}|$ is lower than we found so far: namely 11.

Table 6.6: Computational results of the rolling horizon heuristic, based on decomposition method 3. Parameter k determines the size of M^{sub} and l the size of M^{RH}. First row shows the results of the DSP without decomposition. * : $|M_{\text{unp}}|$. For the gray fields, no experiments are performed.

		C		D		E		F		I		J		K		L	
k	*l*	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**	*	**CPU time**
DSP	→	0	01:39:18	13	10:00:00	0	01:06:56	0	00:10:08	15	10:00:00	13	10:00:00	6	01:33:25	*error*	
3	0	6	00:00:15	16	00:00:25	0	00:23:33	8	00:05:10	18	00:03:30	14	00:02:04	8	00:02:44	3	00:07:25
4	0	0	00:00:15	14	00:00:20	0	00:21:59	4	00:07:11	17	00:03:12	13	00:01:51	7	00:02:44	2	00:07:58
5	0	5	00:00:13	15	00:00:21	0	00:22:35	8	00:05:52	15	00:02:41	13	00:01:43	7	00:03:00	3	00:08:06
6	0	2	00:00:16	14	00:00:23					14	00:02:10	13	00:01:58				
8	0	0	00:01:26	14	00:18:14					15	00:04:45	13	00:02:04				
3	1	1	00:00:20	13	00:00:32	0	00:27:49	2	00:06:42	17	00:02:47	13	00:02:36	8	00:03:36	3	00:10:43
4	1	0	00:00:17	13	00:00:27	0	00:23:28	2	00:05:29	14	00:03:14	14	00:02:06	7	00:03:36	2	00:10:17
5	1	0	00:00:16	13	00:00:36					15	00:02:57	13	00:02:22				
6	1	1	00:00:17	13	00:02:26					10	00:06:34	13	00:02:12				
8	1	0	00:01:32	13	00:08:03					13	00:09:19	14	00:02:36				
3	2	0	00:00:20	14	00:00:41	0	00:35:34	1	00:08:37	12	00:03:33	13	00:03:12	6	00:04:19	3	00:15:28
4	2	0	00:00:31	12	00:00:35	0	00:31:19	2	00:07:34	13	00:03:40	14	00:02:19	7	00:03:58	4	00:15:13
5	2	0	00:00:37	14	00:01:54					13	00:05:47	13	00:02:27				
6	2	0	00:01:22	13	00:24:29					12	00:06:57	13	00:02:47				
8	2	0	00:03:34	13	00:09:04					10	05:47:33	14	00:06:47				
3	4	0	00:01:17	13	00:02:56												
4	4	0	00:04:33	12	00:15:37												
5	4	0	00:01:32	13	00:08:42												
6	4	0	00:02:09	12	00:10:20												
8	4	0	00:52:35	11	00:31:40												

🟩	best solution	🟧	> 1 hour
🟩	2nd best	🟧	> 30 min
🟩	3rd best	🟧	> 5 min

Chapter 7

The extended model for composed movements

Until this point we studied the Shunt Routing Problem as described in Chapter 2. The Shunt Routing Problem considers a shunt yard at the end of the day, where trains are driven from platform to park track. However, the actual situation is a bit more complicated. Trains are driven to their park track, possibly via one or two service locations to undergo maintenance activities. The following morning, trains are routed to their assigned platform track. The route of a train between entering the station at night and leaving the station the following morning can be described by its *composed movement* consisting of several *submovements*.

In this chapter, we make the step towards practical implementation by presenting an extended model for shunt routing of composed movements of trains. We consider the time window between the rush hour in the evening and the rush hour the following morning (i.e. around 8.p.m. - 6 a.m.). In Section 7.1 we start with defining the Composed Shunt Routing Problem. Section 7.2 explains which changes are needed to transform the Disjoint Shortest Paths algorithm (DSP) to an algorithm which can solve the composed problem. We name this algorithm *the Disjoints Shortest Paths algorithm for Composed Movements* (DSP-C). We present results of exemplary experiments on this algorithm in Section 7.3.

7.1 The Composed Shunt Routing Problem

We define the Composed Shunt Routing Problem as follows:

Composed Shunt Routing Problem
Instance: We first list all relevant data and nomenclature, which is explained subsequently.

- The topology of the considered shunt yard, which consists of its tracks and switches.
- A set of (composed) shunting movements $M = M^1 \cup\ M^2 \cup\ \ldots \cup\ M^N$, where M^i $(1 \leq i \leq N)$ denotes the composed movement of train i. Composed movement $M^i = \{m^i_1, m^i_2, \ldots, m^i_{Ni}\}$, where any submovement m^i_j is described by the tuple $(\tau_{\text{start}}, \tau_{\text{finish}}, r, d, \delta_{\text{start}}, \delta_{\text{finish}})$ with τ_{start} : departure track, τ_{finish} : arrival track, r : release, d : deadline, δ_{start} : direction of departure and δ_{finish} : direction of arrival.
- A set of reservations $R = \{(e, t_0, t_f) \mid e :$ infra element (track/switch), t_0 : start reservation, t_f : final time reservation$\}$, which determine the periods that certain infra elements of the shunt yard are not available for shunting movements.

Goal: Finding a schedule where submovements $m \in M$ are assigned an execution time and a route, satisfying the given time windows and assigned parking tracks, avoiding conflicts at the shunt yard and taking into account the reservations. The primary goal is to minimize the number of unplanned submovements. A solution is feasible when all submovements $m \in M$ are planned. Next to feasibility, the secondary objective is to minimize the costs.

Note that this problem description is similar to the definition of the Shunt Routing Problem, introduced in Chapter 2. We limit ourselves to explaining the changed parts. M^i denotes the composed movement of train i, consisting of a set of N_i submovements. A submovement describes a *simple* movement, for example the movement from platform track to service track or from park track to platform track. *Submovements* are similar to *movements* in the Shunt Routing Problem. They are described by 6-tuples $(\tau_{\text{start}}, \tau_{\text{finish}}, r, d, \delta_{\text{start}}, \delta_{\text{finish}})$. The track of departure (τ_{start}) states where the train leaves and the arrival track (τ_{finish}) states the destination of this submovement. A time window $([r, d])$ is given in which this movement needs to be planned. An instance might include information about the directions (A or B) of leaving (δ_{start}) or entering (δ_{finish}) a track. However, when this field is left empty, this direction is not imposed beforehand. Different from the movements in the Shunt Routing Problem, the order at the parking track is not given.

Given submovement m^i_j of train i, we assume $\tau^j_{\text{finish}} = \tau^{j+1}_{\text{start}}$, i.e. train i departs in submovement $j+1$ from the track where it has arrived in the previous submovement, the j^{th}. Furthermore, we assume that there is no overlap in time windows and the submovements are ordered chronologically, i.e. $r^i_j < d^i_j \leq r^i_{j+1} < d^i_{j+1}$ $(1 \leq j \leq N_i$, for any train $i)$.

Figure 7.1 depicts an example of a composed movement consisting of three submovements, $\{m_1, m_2, m_3\}$. The train is assigned to park at track τ_{park}. Before parking, the train has to undergo cleaning at service track τ_{service}. The planning time for this cleaning is given and marked in green. To ensure that the train is at the service track during this time window, the starting point of the cleaning is set as the deadline of the first submovement and the end point of the cleaning as the release of the second submovement.

In Chapter 2 we listed some simplifications for the Shunt Routing Problem, such as the assumption that the shunt yard is empty at the beginning of the considered time horizon. Most of these simplifications also hold for the composed version of the problem. Only the midnight constraint and the availability of parking parts for assigned trains are dropped.

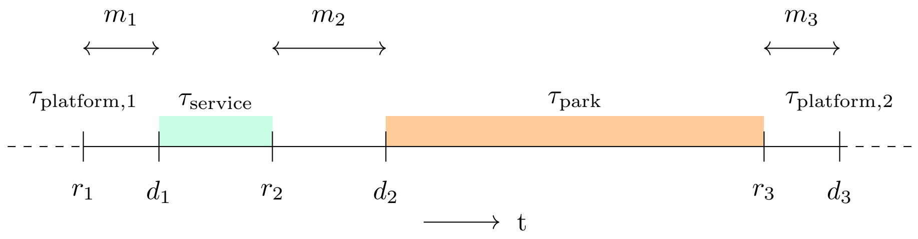


Figure 7.1: A timeline of a composed movement consisting of three submovements, $\{m_1, m_2, m_3\}$. In green a planned service activity and in orange the planned parking time is marked. Releases and deadlines of the submovements are denoted by r_i and d_i respectively.

7.2 Disjoint Paths formulation

Chapter 4 describes the transformation of the Shunt Routing Problem into a Disjoint Paths Problem with additional constraints. An ILP-formulation is given to solve the problem. The main structure of this formulation can be used to solve the Composed Shunt Routing Problem as well. In this section, we list the changes in the algorithm which are needed to transform the Disjoint Shortest Paths algorithm (DSP) to the Disjoint Shortest Paths algorithm for Composed Movements (DSP-C).

- **Splitting park tracks**
  In Section 4.1, we described the construction of the underlying time-expanded network. In the first step, open parking tracks were split into two LIFO tracks. This is possible, because departures from parking tracks are out of consideration in the Shunt Routing Problem. However, in the problem with composed movements, this leads to a loss of quality: when a train enters an open parking track at side A, it might leave the track via side B or vice versa. Therefore open parking tracks are not split in the DSP-C.

  In the DSP, the parking tracks are also split in track segments based on the number of trains which are assigned to park there. When for example three trains are assigned to park at track τ_{park}, this track is split into three parts which are considered as independent infra elements. This splitting cannot simply be adopted by the DSP-C. Assume for example that during one night, three trains are assigned to service track τ_{service} sequentially to undergo a maintenance activity. There is no overlap in the planned service times. Splitting τ_{service} in three infra elements would not be appropriate, since at most one train stays at τ_{service} at the time. Therefore, in the DSP-C we split tracks based on the number of trains present at its busiest moment.

  Figure 7.2 shows an example of a service track τ to illustrate how to split tracks. The periods of service of the four movements assigned to τ are marked in green and the time windows of the submovements before are depicted. At the busiest moment, two trains are present at the track. Therefore, in this example τ is split into two parts.

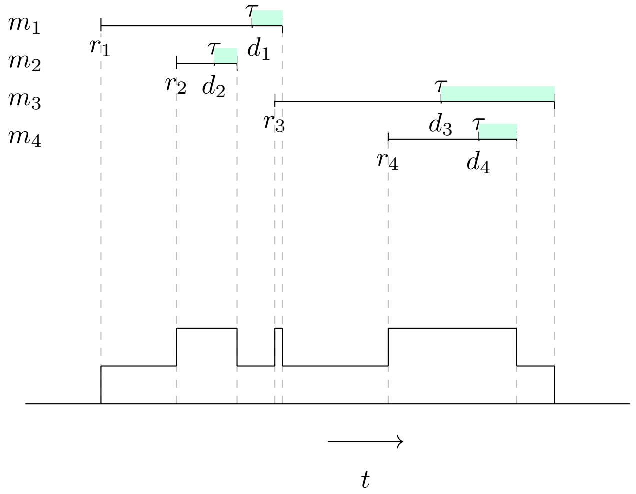


Figure 7.2: Example illustrating how to determine the number of parts of track τ: there are at most two trains at the time. For every submovement, we see the moment from release to drive to $\tau_{\text{finish}} = \tau$ until the release of the following submovement.

- **Sources and targets**
  A vertex in the time-expanded network corresponds to an infra element at a specific point in time at a specific direction. In the DSP, trains are assigned a specific order at the parking track and therefore movements are assigned to a specific part of the (split) park track. In the composed

model, a train can be routed to any of the parts of the track of arrival. Similarly, a train can depart from any part of the departure track. It is up to the algorithm to determine conflict free orders at the tracks. Therefore, sources and targets for submovements are linked to any of these track parts.

To ensure that a train which arrived at part k of the track, departs from the same part in the subsequent submovement, we need to add constraints to the ILP. For any submovement m_j with $j < N$ and $\tau_{\text{finish}} = \tau$ where τ is any track split in multiple parts, for every part k of track τ, we need to add the following constraint to the model:

$$x^{m_j}_{u_1,t_{m_j}} + x^{m_j}_{u_2,t_{m_j}} \leq x^{m_{j+1}}_{s_{m_{j+1}},u_3} + x^{m_{j+1}}_{s_{m_{j+1}},u_4} \ , \tag{7.1}$$

where

$$\begin{aligned} u_1 &= v_{\tau(k),d_{m1},A} \\ u_2 &= v_{\tau(k),d_{m1},B} \\ u_3 &= v_{\tau(k),r_{m2},A} \\ u_4 &= v_{\tau(k),r_{m2},B} . \end{aligned}$$

Here $\tau(k)$ denotes the k^{th} part of track τ. Constraint (7.1) ensures that, when submovement m_j enters its target t_{m_j} at part k in any direction, the subsequent submovement of this composed movement needs to depart from the same part in any direction.

- **Blocks between submovements**
In Figure 7.1 we presented an example of a composed movement, consisting of three submovements. In between these submovements, the train stays at respectively a service track (marked in green) and a parking track (marked in orange). The schedule which determines the service and parking times is given (in OPG+, this is determined in previous submodules). We are left to block these infra elements during these periods in between the submovements to avoid other trains using these infra elements simultaneously. However the difficulty is that for split tracks, we do not know beforehand which part to block. Therefore it is not possible to simply add a reservations to R. We introduce a new type of constraint to block the infra elements between two subsequent submovements.

For any pair of subsequent submovements $m^i_j, m^i_{j+1} \in M^i$, for all parts k of $\tau^j_{\text{finish}} = \tau$, for all $t \in (d_{m_j}, r_{m_{j+1}})$ in the chosen granularity, we add the following constraint:

$$\sum_{\delta \in \{A,B\}} x^{m^i_j}_{u_1,t_m} + \sum_{\tilde{m} \in M \setminus M^i} \ \sum_{\tilde{v} \in \text{OUT}(u_2)} \ \sum_{\delta \in \{A,B\}} x^{\tilde{m}}_{\tilde{v},u_2} \leq 1 \ , \tag{7.2}$$

where

$$\begin{aligned} u_1 &= v_{\tau(k),d_{m^i_j},\delta} \\ u_2 &= v_{\tau(k),t,\delta} . \end{aligned}$$

The outgoing edges of node v are reffered to as OUT(v). Constraint (7.2) ensures that, whenever submovement m^i_j enters its track of arrival at part k ($\tau(k)$), this track part cannot be used by any other submovement $\tilde{m} \in M \setminus M^i$ until the release of the subsequent submovement m^i_{j+1}. Figure 7.3 illustrates this constraint. When submovement m^i_j enters its track of arrival at part k, the red marked nodes in the time-expanded network cannot be used by any other submovement.

- **Removal Constraints** (4.8) **and** (4.7)
From the ILP formulation presented in Section 4.2 we remove the two non-fundamental constraints. Constraint (4.7) ensures that parking tracks can be used by parking trains only. However, the arrival tracks of submovements (similar to the parking tracks for movements) are not necessarily parking tracks and therefore must be available for other trains as well. A submovement can for example

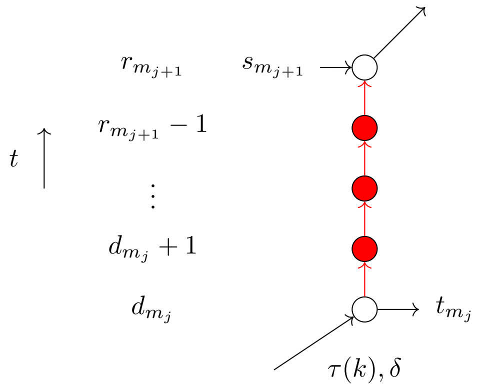


Figure 7.3: Illustration of Constraint (7.2): when movement m_j arrives at $\tau(k)$, the red nodes cannot be used by any other submovement.

describe the movement back to the platform track. It would lead to problems if we restrict usage of this platform track to trains arriving there.

Constraint (4.8) enforces that once a train enters its parking track, it is forced to drive to its assigned position and stays there until the end of the considered time horizon. First of all, trains in the composed model are not assigned a specific position at the track. Above that, trains must be able to leave their track of arrival in the subsequent submovement and therefore this constraint needs to be removed.

7.3 Computational results

To test the DSP-C, we perform some exemplary experiments, which we present in this section. The settings of the experiments, as well as the data sets of the topology of the stations, the CPLEX version and computer are the same as in the earlier experiments of Chapter 5. We compare the results of the DSP-C with the RouteZoeker algorithm (RZ, see Section 5.2).

We created a set of four instances to the Composed Shunt Routing Problem of which one is on station Enschede and three are on station Eindhoven. The characteristics of these instances are shown in Table 7.1. Data sets O and R are based on sets of real data. They describe all movements at the shunt yard during one night. Instances P and Q are artificially created instances. Q is an unrealistically dense set of movements ($|M| = 24$ and the considered time horizon is less than an hour), created to challenge the algorithms. The details of the instances can be found in the Appendix in Tables 8.15 - 8.18.

Table 7.1: Characteristics of the considered instances. $|M|$ denotes the number of submovements to be planned, $\overline{d_m - r_m}$ denotes the average length of the time windows of the movements ([hh:mm:ss]), $|V|$ and $|E|$ denote the number of nodes respectively edges in the underlying time-expanded graph.

Set	**Location**	$\vert M\vert$	$\vert V\vert$	$\vert E\vert$	**Time horizon**	$\overline{d_m - r_m}$
O	Enschede	25	582022	847726	18h04 - 08h00	0:59
P	Eindhoven	9	377356	556488	18h30 - 20h15	0:15
Q	Eindhoven	24	186668	280443	18h30 - 19h20	0:15
R	Eindhoven	60	3047430	4671427	18h16 - 07h42	0:24

Statement 7: Similar to the DSP, the DSP-C finds better routing schedules compared to RZ. For the largest instances, heuristics are needed to speed up the algorithm and avoid memory errors.

Table 7.2: Computational results of the benchmark on DSP-C and RZ. The objective value is denoted by σ and $|M_{\text{unp}}|$ denotes the number of movements for which no route is found. $t_{\max}$ denotes the parameter of the shifting deadlines heuristic, when applied. Results of DSP-C are marked green if the result is strictly better than the result of RZ and red if the result is strictly worse. [1]: computation of DSP-C is stopped prematurely.

		DSP-C			**RZ**		
Set	$t_{\mathbf{max}}$	σ	$\lvert M_{\mathbf{unp}}\rvert$	**CPU time**	σ	$\lvert M_{\mathbf{unp}}\rvert$	**CPU time**
				[hh:mm:ss]			[hh:mm:ss]
O	-	210000	0	00:39:51	10190000	1	00:43:57
O	60 min	210000	0	00:13:15	10190000	1	00:12:26
P	-	10070000	1	00:02:27	40040000	4	00:03:49
Q[1]	-	110130510	11	00:15:00	70173380	7	02:48:00
		30212200	3	10:00:00			
R	-	*error*			*error*		
R	10 min	*error*			*error*		

The results of the experiments are shown in Table 7.2. For the two instances based on real data, O and R, we also performed experiments where the shifting deadlines heuristic is applied. Results for instance R are missing, because the memory needed for the ILP exceeded the limits of CPLEX. Even when we eliminated decision variables by the shifting deadlines heuristic, the memory error occurs. Note that Eindhoven is one of the biggest shunt yards in the Netherlands. For applying the DSP-C to this location, (more) heuristics are needed.

The green-marked results of the DSP-C are strictly better than the results of RZ and the red-marked result is strictly worse. For small instance P, we observe the potential of the DSP-C. Here $|M^{\text{RZ}}_{\text{unp}}| = 4$, where $|M^{\text{DSP-C}}_{\text{unp}}| = 1$. The potential of the DSP-C is also shown by the experiment on instance C, which is the data set containing all movements during one night at station Enschede. RZ does not succeed in finding a feasible schedule for this instance ($|M^{\text{RZ}}_{\text{unp}}| = 1$). However, the DSP-C finds a feasible schedule within 40 minutes. When we apply the shifting deadlines heuristic with upperbound $t_{\max} = 60$ minutes, DSP-C finds a feasible solution in only 13 minutes.

The experiment of the DSP-C on Q is stopped prematurely after ten hours of running time. The best found solution after this time is outperforming RZ with $|M^{\text{DSP-C}}_{\text{unp}}| = 3$ compared to $|M^{\text{RZ}}_{\text{unp}}| = 7$. In Table 7.2, we also present the best solution found within the first 15 minutes of computation. In this solution, marked in red, 11 movements are left unplanned.

Chapter 8

Conclusions and discussion

The starting point of this study was the tool OPG+ developed by NS to support shunt planners. We zoomed in to the routing problem and defined the Shunt Routing Problem, which is to find routes and planning times for movements from platform track to park track. We modeled the Shunt Routing Problem as a Disjoint Paths Problem on a time-expanded network. A path through the network corresponds to a route through the infrastructure of a shunt yard in terms of location, time and direction. Additional constraints of a forbidden pair-form are needed to model the problem properly. Both, the Disjoint Paths Problem and the Impossible Pair constrained Path problem are NP-complete. Therefore, we formulated our model as an Integer Linear Program and solved the problem using CPLEX.

The main advantage of this Disjoint Paths approach, compared to the method currently implemented in NS-tool OPG+, is that it allows more freedom in finding a proper routing schedule. Routes are not fixed beforehand and the option of waiting is introduced. Using a time-expanded network is an intuitive way to model the Shunt Routing Problem. Besides this, it allows us to model practical things such as other rolling stock on the shunt yard and periods that infra elements are out of use. Orderings at parking tracks are taken into account. Another advantage of this model is that it is applicable to any shunt yard in general.

To investigate the performance of the algorithm, we executed experiments on twelve instances based on station Enschede and Eindhoven. These instances are partly based on real data and partly artificially created to challenge our algorithm. We compared the performance of our Disjoint Shortest Paths algorithm (DSP) to the performance of two other methods: the greedy Successive Shortest Paths algorithm (SSP) and the RouteZoeker algorithm (RZ) which is based on the submodule of OPG+.

Table 5.3 shows the promising results of these experiments. In all experiments, SSP is outperformed by DSP and in most cases RZ as well. The experiments confirmed that, especially when the problem becomes very challenging, the extra freedom of the DSP pays off. In those experiments, DSP finds a schedule where more movements can be executed than in the schedule created by RZ. For example, in the experiment on an artificially created, -unrealistically- dense set of movements, DSP finds a schedule where all movements can be executed, where in the schedule of RZ only half of the movements are included in the schedule. However, the freedom of DSP has its price: the algorithm needs in most experiments a longer computation time than RZ. In some experiments the computation time explodes and on the largest instance, the memory limits of CPLEX are exceeded.

To speed up the algorithm and solve the memory problem, we investigated various heuristic methods. In a *shifting deadlines* heuristic, we introduce an upper bound, $t_{\max}$, for the length of a time window in which a movement can be executed. The heuristic method forces movements to be planned in the first $t_{\max}$ minutes of their available time window. This shifting deadlines heuristic results in significant improvements in computation times hardly loosing quality of the solution. In the experiments, the computation time is reduced with a factor around 10.

Another heuristic is the *decomposition algorithm*, which splits the set of movements in smaller instances and solves these subproblems sequentially. We investigated various ways of selecting these subsets of movements. Furthermore, we applied a *rolling horizon algorithm*, which in addition also takes into account some extra movements in the future or past when solving the subproblem. There is no decomposition or rolling horizon method that stands out because of consistently good performance. The performance of the different methods strongly depends on the instance. Overall we oberved that decomposition and rolling horizon methods find solutions of good quality in a significantly smaller amount of time. More research is needed to investigate which method would be best applicable for NS.

To make a step towards practical implementation of the Disjoint Paths approach, we extended our algorithm for *composed movements*. A composed movement of a train describes all submovements of a train between entering and leaving the shunt yard. The main structure of the DSP could be used to build the DSP-C (Disjoint Shortest Paths approach for Composed movements), mainly by splitting composed movements in submovements. Some exemplary experiments showed that the DSP-C, similar to the DSP, finds better routing schedules compared to RZ. For the largest instances, heuristics are needed to reduce the computation time and avoid memory errors.

8.1 Practical recommendations to NS

The Disjoint Paths approach might be a valuable alternative for submodule RouteZoeker of OPG+. Especially when the *shunting puzzle* becomes very challenging, the freedom of this method outperforms RouteZoeker in finding feasible solutions. We implemented an extended model for composed movements, which comes closer to reality. Combining this model with heuristics might be the way to find high quality solutions within a short amount of time. For implementation the following things need to be considered:

- Avoiding long computation times and errors because of exceeded memory capacities requires more research. Experiments on real data sets of different locations are needed. In this study, we investigated different heuristic methods. Shifting deadlines appeared to be an easy and effective way to reduce the computation time without loosing quality of the solution. For other (decomposition or rolling horizon) heuristics, more experiments are needed to conclude which method performs best. In Section 8.2 we propose some ideas for other heuristic methods which can be investigated. It is recommended to consider a combination of different heuristic methods.
- Some practical things need to be incorporated in the algorithm, such as preference routes and norms for crossings. The norms can be incorporated by adding new constraints to the ILP to introduce a required idle time before and after the moment the crossing is used. Preference routes can be taken into account by introducing weights ω_e^m to penalize movement m when crossing infra element e.
- In this study, we considered identical trains only. It is relatively easy (i.e. solely administrative) to take into account the different train types, to for example vary driving times or restrict tracks for specific train types.
- We assumed that the topology is empty at the beginning of the time horizon. This can easily be changed by introducing reservations to block the infra elements which are already in use.
- Coupling and decoupling of trains was outside the scope of this research. When this (de)coupling activities are planned beforehand, this can be incorporated in the model for composed movements, by adding an extra submovement to the location of (de)coupling.
- For the crew planning, it is desirable to minimize the number of parallel planned operations to minimize the crew size. This can be included in the ILP formulation in several ways. It is for example possible to include weights in the objective function to penalize parallellity or to add a constraint which makes it impossible to plan more than k operations simultaneously. It is also possible to implement a combination of these two things.

8.2 Suggestions for further research

We list the following suggestions for further research:

- In our model we assumed that the assignment of parking tracks is fixed. However, an unplanned movement might be inserted in the schedule by changing the assigned destination. An idea for further research is to investigate a model where changing the assigned (parking) track is allowed. Similarly, in the model for composed movements we assumed that the service planning is fixed, which can be relaxed so that service locations and times can be changed.
- The Shunt Routing Problem can easily be redefined to an incremental version, where N movements are already scheduled and the $(N+1)^{\text{th}}$ movement needs to fit in the existing schedule. The goal can be to minimize the adaptions in the existing schedule. Solving this incremental version of the shunting problem can be interesting for two purposes: 1) as a replanning tool for last minute disruptions and 2) as a fast heuristic for building a shunt plan.
- In Chapter 4.4 we discussed the choice of a proper time grid underlying the time-expanded network. In our experiments, we used a relatively easy time discretization with time steps of 10 seconds during the whole time horizon for every infra element. For further research, we suggest to investigate the effect of changing the time discretization on the computation time. More advanced granularities can be tested to speed up the algorithm, for example using a sparse time grid in the *easy* parts of the time-expanded network and very dense time grid in the *difficult* parts.
- We suggest to investigate the option of allowing delay in the solution (similar as in the master thesis of Van Den Broek (2016)), in the philosophy that a delayed movement is preferred over an unplanned movement. The goal can be redefined as minimizing the total delay.
- In our approach, lengths of trains are not taken into account. We assume that in the assignment of the (parking) tracks the length restrictions are satisfied. We modeled the train as a point in space. However, it would be more realistic to model a train as a *line* in space. For implementation it is needed to investigate whether this simplification leads to conflicts in real life. Furthermore, it might be interesting to think of a model where trains are modeled as lines.
- We suggest to experiment with different existing heuristics for solving ILP's quickly, for example column generation. Another idea is to run CPLEX with a *warm start*, i.e. starting with an initial solution, generated by a quick method (for example SSP). It might also be relevant to consider other ILP solvers, and use more capable computational resources for the solution of large ILPs.

Bibliography

Ahuja, R.K., T.L. Magnanti, and J.B. Orlin (1993). “Network flows: theory, algorithms and applications.” In: *Prentice Hall, Englewood Cliffs, New Jersey.*

Bettinelli, A., A. Santini, and D. Vigo (2017). “A real-time conflict solution algorithm for the train rescheduling problem”. In: *Transportation Research Part B* 106, pp. 237–265. DOI: 10.1016/j.trb.2017.10.005.

Blasum, U. et al. (2000). “Scheduling trams in the morning”. In: *Mathematical Methods of Operations Research* 49 (1), pp. 137–148. DOI: 10.1007/PL00020912.

Burggraeve, S. (Sept. 2017). “Passenger robust timetables for dense railway networks”. PhD thesis. Arenberg Doctoral School, Faculty of Engineering Science.

Cacchiani, V., A. Caprara, and P. Toth (2010). “Scheduling extra freight trains on railway networks”. In: *Transportation Research Part B: Methodological* 44 (2), pp. 215–231. DOI: 10.1016/j.trb.2009.07.007.

Den Hartog, M.R. (2010). *Shunt planning, an integral approach of matching, parking and routing.*

Freling, R. et al. (2005). “Shunting of passenger train units in a railway station”. In: *Transportation Science* 39 (2), pp. 261–272. DOI: 10.1287/trsc.1030.0076.

Fuchsberger, M. (2007). “Solving the train scheduling problem in a main station area via a resource constrained space-time integer multi-commodity flow”. MA thesis. ETH Zurich: Institute for Operations Research.

Gabow, H.N. (1976). “On Two Problems in the Generation of Program Test Paths”. In: *IEEE Transactions on Software Engineering* SE-2 (3), pp. 227–231. DOI: 10.1109/TSE.1976.233819.

Gallo, G. and F. Di Miele (2001). “Dispatching buses in parking depots”. In: *Transportational Science* 35 (3), pp. 322–330. DOI: 10.1287/trsc.35.3.322.10151.

Haahr, J.T., R.M. Lusby, and J.C. Wagenaar (2017). “Optimization methods for the Train Unit Shunting Problem”. In: *European Journal of Operations Research* 262. DOI: 10.1016/j.ejor.2017.03.068.

Haijema, R., C.W. Duin, and N.M. Van Dijk (2006). “Train shunting: A practical heuristic inspired by dynamic programming”. In: the Netherlands: Wiley Online Library. Chap. 16, pp. 437–475. DOI: 10.1002/0471781266.ch16.

Hamdouni, M., G. Desaulniers, and F. Soumis (2004). “Parking buses in a depot using block patterns: a Benders decomposition approach for minimizing type mis-matches”. In: *Computers and Operations Research, Canada.* DOI: 10.1016/j.cor.2006.02.002.

Hermann, T. and G. Caimi (2006). “Model and algorithm for rerouting delayed trains online”. In: *Euro XXI Conference.*

Jacobsen, P.M. and D. Pisinger (2011). “Train shunting at a workshop area”. In: *Flexible Services and Manufacturing Journal* 23 (2), pp. 156–180. DOI: 10.1007/s10696-011-9096-1.

Jiao, F. and S. Dong (2018). “Ordered Escape Routing with Consideration of Differential Pair and Blockage”. In: *ACM Trans. Des. Autom. Electron. Syst.* 23 (4).

Karp, R.M. (1975). “On the computational complexity of combinatorial problems”. In: *Networks, an International Journal* 5 (1), pp. 45–68. DOI: 10.1002/net.1975.5.1.45.

Kawarabayashi, K., Y. Kobayashi, and B. Reed (2012). “The disjoint paths problem in quadratic time”. In: *Journal of Combinatorial Theory* 102, pp. 424–435. DOI: 10.1016/j.jctb.2011.07.004.

Kroon, L.G., R.M. Lentink, and A. Schrijver (2008). “Shunting of Passenger Train Units: an Integrated Approach”. In: *Transportation Science* 42 (4), pp. 405–549. DOI: 10.1287/trsc.1080.0243.

Lee, Y. and N. Y. Hsu (2007). “An optimization model for the container pre-marshalling problem”. In: *Computers & Operations Research* 34, pp. 3295–3313. DOI: 0.1016/j.cor.2005.12.006.

Lentink, R.M. (Feb. 2006). “Algorithmic Decision Support for Shunt Planning”. PhD thesis. Erasmus School of Economics Erasmus University Rotterdam.

Lynch, J.F. (1975). “The equivalence of theorem proving and the interconnection problem”. In: *ACM SIGDA Newsletter* 5, pp. 31–65.

NOS (2018). *Het spoor begint vol te raken.* URL: nos.nl/artikel/2246806-prorail-het-spoor-begint-vol-te-raken.html (visited on 08/20/2018).

ProRail (2018). *ProRail in cijfers.* URL: www.prorail.nl/over-prorail/wat-doet-prorail/prorail-in-cijfers (visited on 10/18/2018).

Robertson, N. and P.D. Seymour (1995). “Graph Minors .XIII. The Disjoint Paths Problem”. In: *Journal of Combinatorial Theory, Series B* 63 (1), pp. 65–110. DOI: 10.1006/jctb.1995.1006.

Schrijver, A. (2003). *Combinatorial Optimization.* Part VII, Chapter 70. the Netherlands.

Tomii, N. and L.J. Zhou (2000). “Depot shunting scheduling using combined genetic algorithm and PERT.” In: *Computer in Railways VII* 50, pp. 437–446. DOI: 10.2495/CR000421.

Tomii, N., L.J. Zhou, and N. Fukumara (1999). “An algorithm for station shunting scheduling problems combining probabilistic local search and PERT.” In: *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems.* Springer, pp. 788–797. DOI: 10.1007/978-3-540-48765-4_84.

Van Den Broek, J. (Dec. 2009). “MIP-based Approaches for Complex Planning Problems”. PhD thesis. Technische Universiteit Eindhoven. DOI: 10.6100/IR653241.

Van Den Broek, R. (2016). “Train Shunting and Service Scheduling: an integrated local search approach”. MA thesis. the Netherlands: Universiteit Utrecht.

Winter, T. (1999). “Online and real-time dispatching problems”. PhD thesis. Germany: Technical University, Braunschweig.

Winter, T. and U.T. Zimmermann (2000). “Real-time dispatch of trams in storage yards”. In: *Annuals of Operations Research* 96, pp. 287–315. DOI: 10.1023/A:1018907720194.

Wolfhagen, F.E. (2017). “The Train Unit Shunting Problem with Reallocation”. MA thesis. the Netherlands: Erasmus University Rotterdam.

Zwaneveld, P.J. et al. (1996). “Routing Trains Through Railway Stations: Model Formulation and Algorithms”. In: *Transportation Science* 30 (3), pp. 181–196.

Appendix

Station Enschede

Table 8.1: Characteristicss of the tracks of station Enschede. The second column denotes the time to traverse the tracks (based on the length). ω_{wait} and ω_{saw} denote the weights for using this track for waiting respectively sawing.

	time	length		
name	[sec]	[m]	ω_{wait}	ω_{saw}
405a	50	451	10	10000
412	20	152	10	50000
411	20	152	10	50000
408	40	349	0	10000
407	40	344	0	10000
406	60	508	0	10000
405b	60	513	0	10000
415	30	207	1	10000
414	40	330	1	10000
404a	20	192	0	10000
403	30	256	0	10000
402	40	352	0	10000
401	40	354	0	10000

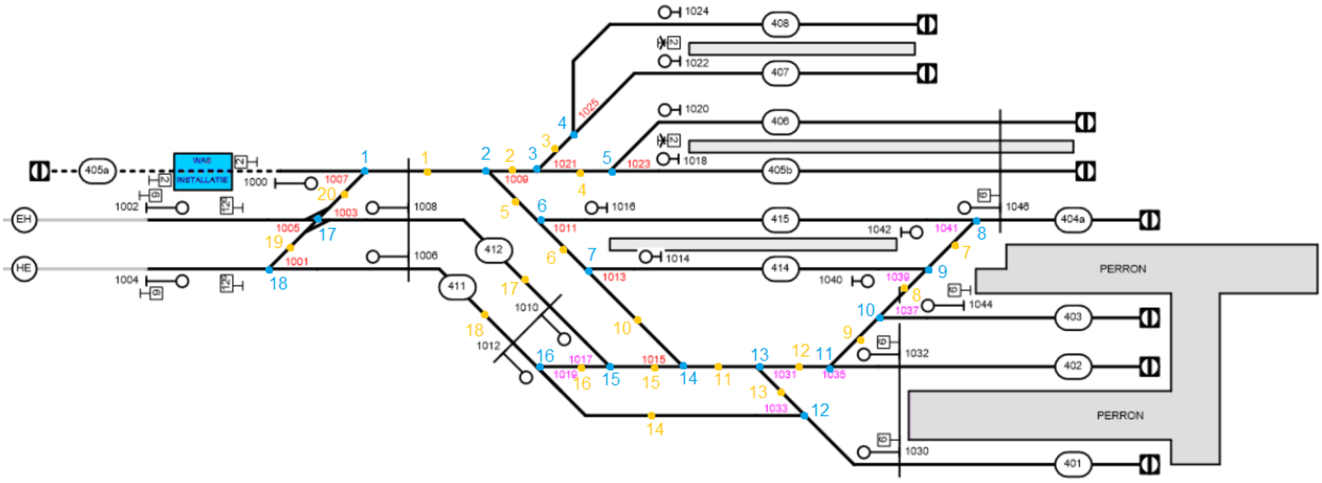


Figure 8.1: Infrastructure of station Enschede.

Station Eindhoven

Table 8.2: Characteristicss of the tracks of station Eindhoven. The second column denotes the time to traverse the tracks (based on the length). ω_{wait} and ω_{saw} denote the weights for using this track for waiting respectively sawing. When the value of the last column is empty, it is not allowed to use this track as sawing track.

	time	**length**				**time**	**length**		
name	**[sec]**	**[m]**	ω_{wait}	ω_{saw}	**name**	**[sec]**	**[m]**	ω_{wait}	ω_{saw}
40	40	322	10	10000	41	30	204	0	10000
39	50	475	1		22	10	29	0	
38	50	477	1		21	30	299	0	10000
37	50	481	1		46	40	337	0	10000
36	50	485	1		45	40	382	0	10000
35a	30	265	10	10000	44	50	425	0	10000
7	70	690	1		43	40	375	0	
6	40	353	10	10000	62	10	58	0	
5	40	395	10	10000	67	90	829	0	
4	40	342	10	10000	66	60	589	0	
3	40	344	10	10000	65	60	570	0	
2	50	401	10	10000	42b	20	106	0	10000
1	40	327	10	10000	156a	20	162	0	
18a	20	182	0		157a	20	151	0	
35b	20	105	0		153	20	122	0	
17	60	567	0		155	10	72	0	
16	50	475	0		64	70	691	0	
15	50	475	0		63	40	400	0	
34	20	126	0		61	50	488	0	20000
31	10	65	0		164	10	19	0	
14	30	217	0		165	10	75	0	
13	30	217	0		158	10	100	0	
12	30	209	0		47	20	129	0	10000
11	30	208	0		161	30	245	0	
32	10	28	0		162	20	136	0	
18b	30	294	0		163	20	141	0	
133	60	546	0		157b	20	161	0	
132	40	328	0		156b	20	126	0	
131	40	328	0		157c	10	81	0	
130	40	328	0		173	20	195	0	
129	40	328	0		172	20	191	0	
50	10	53	0		166	30	270	10	10000
42a	20	163	0	10000	71	10	100	0	

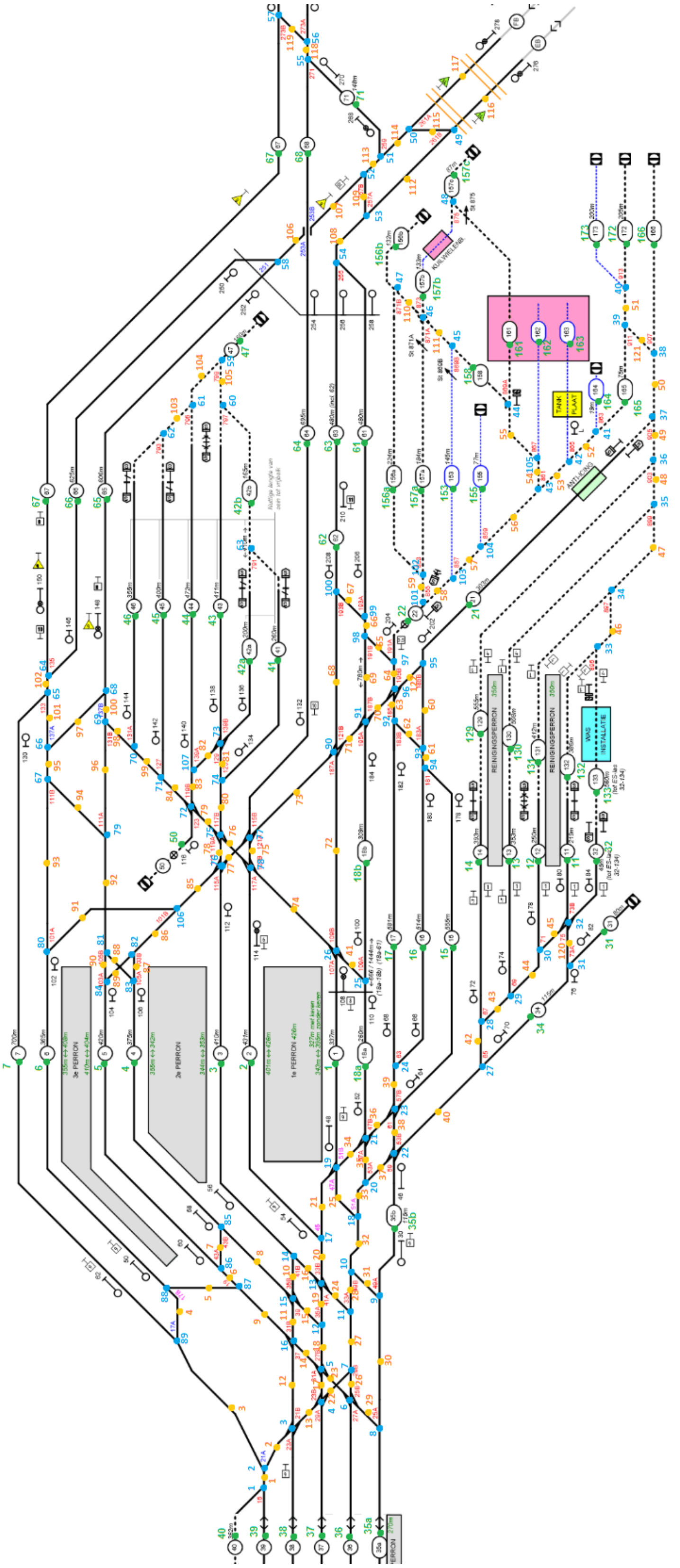


Figure 8.2: Infrastructure of station Eindhoven.

Instances benchmark

Table 8.3: Characteristics of M_A, instance for location Enschede.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	404a	408	20:00	20:10	A
2	403	408	20:01	20:10	A
3	402	408	20:02	20:10	A
4	401	407	20:02	20:10	A
5	404a	407	20:03	20:10	A
6	402	407	20:05	20:10	A
7	401	406	20:08	20:20	A
8	403	406	20:11	20:20	A
9	402	406	20:12	20:20	A
10	401	414	20:10	20:20	A
11	404a	414	20:13	20:20	B
12	403	415	20:10	20:20	A
13	404a	415	20:15	20:20	B

Table 8.4: Characteristics of M_B, instance for location Enschede.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	404a	405b	20:00	20:10	A
2	403	406	20:01	20:10	A
3	402	407	20:02	20:10	A
4	401	408	20:02	20:10	A
5	404a	406	20:05	20:10	A
6	402	414	20:05	20:10	A
7	401	405b	20:12	20:20	A
8	403	408	20:11	20:20	A
9	402	407	20:12	20:20	A
10	401	408	20:12	20:20	A
11	404a	406	20:15	20:20	A
12	403	414	20:15	20:20	A

Table 8.5: Characteristics of M_C, instance for location Enschede.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	404a	408	20:00	20:30	A
2	403	408	20:00	20:30	A
3	402	408	20:00	20:30	A
4	401	408	20:00	20:30	A
5	404a	407	20:05	20:30	A
6	403	407	20:05	20:30	A
7	402	407	20:05	20:30	A
8	401	407	20:05	20:30	A
9	404a	406	20:10	20:30	A
10	403	406	20:10	20:30	A
11	402	406	20:10	20:30	A
12	401	406	20:10	20:30	A
13	404a	405b	20:15	20:30	A
14	403	405b	20:15	20:30	A
15	402	405b	20:15	20:30	A
16	401	405b	20:15	20:30	A

Table 8.6: Characteristics of M_D, instance for location Enschede.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	404a	408	20:00	20:30	A
2	403	408	20:00	20:30	A
3	402	408	20:00	20:30	A
4	401	408	20:00	20:30	A
5	404a	407	20:02	20:30	A
6	403	407	20:02	20:30	A
7	402	407	20:02	20:30	A
8	401	407	20:02	20:30	A
9	404a	406	20:04	20:30	A
10	403	406	20:04	20:30	A
11	402	406	20:04	20:30	A
12	401	406	20:04	20:30	A
13	404a	405b	20:06	20:30	A
14	403	405b	20:06	20:30	A
15	402	405b	20:06	20:30	A
16	401	405b	20:06	20:30	A
17	404a	408	20:08	20:30	A
18	403	408	20:08	20:30	A
19	402	408	20:08	20:30	A
20	401	408	20:08	20:30	A
21	404a	407	20:10	20:30	A
22	403	407	20:10	20:30	A
23	402	407	20:10	20:30	A
24	401	407	20:10	20:30	A
25	404a	406	20:12	20:30	A
26	403	406	20:12	20:30	A
27	402	406	20:12	20:30	A
28	401	406	20:12	20:30	A
29	404a	405b	20:14	20:30	A
30	403	405b	20:14	20:30	A
31	402	405b	20:14	20:30	A
32	401	405b	20:14	20:30	A

Table 8.7: Characteristics of M_E, instance for location Enschede.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	401	405b	18:19	19:10	A
2	401	405b	18:27	19:33	A
3	402	405b	18:49	19:40	A
4	402	405b	18:57	20:02	A
5	401	405b	19:19	20:06	A
6	401	405b	19:28	20:14	A
7	401	407	20:19	21:10	A
8	401	407	20:29	21:35	A
9	402	407	20:49	21:40	A
10	402	407	20:57	22:03	A
11	401	408	21:19	22:10	A
12	401	408	21:27	22:33	A
13	401	415	22:19	23:10	A
14	401	415	22:27	23:33	A
15	402	407	22:49	23:40	A
16	402	407	22:58	00:04	A
17	401	405b	23:28	00:34	A
18	401	405b	23:19	00:10	A
19	402	415	23:49	00:40	A
20	402	415	23:57	01:03	A
21	404a	415	00:01	00:23	B
22	401	408	00:19	05:13	A
23	401	408	00:27	05:21	A
24	402	405b	00:49	05:54	A
25	402	405b	00:57	06:17	A
26	403	408	01:19	05:11	A
27	403	408	01:28	05:20	A

Table 8.8: Characteristics of M_F, instance for location Enschede.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	401	405b	18:19	19:10	A
2	401	405b	18:20	19:26	A
3	402	405b	18:34	19:35	A
4	402	405b	18:37	19:42	A
5	401	405b	18:49	19:36	A
6	401	405b	18:48	19:34	A
7	401	407	19:19	20:10	A
8	401	407	19:29	20:35	A
9	402	407	19:39	20:30	A
10	402	407	19:47	20:53	A
11	401	408	19:59	20:50	A
12	401	408	20:07	20:13	A
13	401	415	20:19	21:10	A
14	401	415	20:27	21:33	A
15	402	407	20:49	21:40	A
16	402	407	20:48	22:54	A
17	401	405b	21:08	22:14	A
18	401	405b	21:19	22:10	A
19	402	415	21:29	22:20	A
20	402	415	21:37	22:43	A
21	404a	415	21:31	21:53	B

Table 8.9: Characteristics of M_G, instance for location Enschede.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	7	67	20:00	20:15	A
2	6	66	20:00	20:15	A
3	5	63	20:00	20:15	A
4	4	14	20:00	20:15	A
5	3	13	20:00	20:15	A
6	2	12	20:00	20:15	A
7	1	11	20:00	20:15	A
1	7	42b	20:05	20:20	A
2	6	41	20:05	20:20	A
3	5	43	20:05	20:20	A
4	4	45	20:05	20:20	A
5	3	46	20:05	20:30	A
6	2	156a	20:05	20:20	A
7	1	162	20:05	20:20	A

Table 8.10: Characteristics of M_H, instance for location Eindhoven.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	7	67	18:30	19:15	A
2	6	66	18:40	19:05	A
3	5	63	18:48	19:21	A
4	4	14	18:51	19:07	A
5	3	13	18:57	20:15	A
6	7	67	19:40	19:54	A
7	6	66	18:20	19:27	A
8	5	63	19:25	19:40	A
9	4	14	18:40	19:22	A
10	3	13	18:05	19:58	A

Table 8.11: Characteristics of M_I, instance for location Eindhoven.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	1	14	20:00	20:30	A
2	2	14	20:00	20:30	A
3	3	14	20:00	20:30	A
4	4	14	20:00	20:30	A
5	5	14	20:00	20:30	A
6	6	14	20:00	20:30	A
7	7	14	20:00	20:30	A
8	1	13	20:05	20:30	A
9	2	13	20:05	20:30	A
10	3	13	20:05	20:30	A
11	4	13	20:05	20:30	A
12	5	13	20:05	20:30	A
13	6	13	20:05	20:30	A
14	7	13	20:05	20:30	A
15	1	12	20:10	20:30	A
16	2	12	20:10	20:30	A
17	3	12	20:10	20:30	A
18	4	12	20:10	20:30	A
19	5	12	20:10	20:30	A
20	6	12	20:10	20:30	A
21	7	12	20:10	20:30	A
22	1	11	20:15	20:30	A
23	2	11	20:15	20:30	A
24	3	11	20:15	20:30	A
25	4	11	20:15	20:30	A
26	5	11	20:15	20:30	A
27	6	11	20:15	20:30	A
28	7	11	20:15	20:30	A

Table 8.12: Characteristics of M_J, instance for location Eindhoven.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	1	14	20:00	20:30	A
2	2	14	20:02	20:30	A
3	3	14	20:04	20:30	A
4	4	14	20:06	20:30	A
5	5	14	20:08	20:30	A
6	6	14	20:10	20:30	A
7	7	14	20:12	20:30	A
8	1	13	20:02	20:30	A
9	2	13	20:04	20:30	A
10	3	13	20:06	20:30	A
11	4	13	20:08	20:30	A
12	5	13	20:10	20:30	A
13	6	13	20:12	20:30	A
14	7	13	20:14	20:30	A
15	1	12	20:04	20:30	A
16	2	12	20:06	20:30	A
17	3	12	20:08	20:30	A
18	4	12	20:10	20:30	A
19	5	12	20:12	20:30	A
20	6	12	20:14	20:30	A
21	7	12	20:16	20:30	A
22	1	11	20:06	20:30	A
23	2	11	20:08	20:30	A
24	3	11	20:10	20:30	A
25	4	11	20:12	20:30	A
26	5	11	20:14	20:30	A
27	6	11	20:16	20:30	A
28	7	11	20:18	20:30	A

Table 8.13: Characteristics of M_K, instance for location Eindhoven.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	2	45	18:16	18:26	A
2	6	131	18:16	18:26	A
3	2	45	18:16	18:26	A
4	3	133	18:17	18:24	B
5	1	129	18:17	18:24	B
6	6	130	18:23	18:37	B
7	3	132	18:23	18:37	A
8	5	133	18:23	18:37	B
9	2	43	18:18	18:36	A
10	3	132	18:18	18:36	A
11	35a	133	18:18	18:36	B
12	6	132	18:18	18:36	B
13	2	45	18:20	18:40	A
14	3	43	18:20	18:40	A
15	2	42a	18:36	18:46	A
16	3	131	18:36	18:50	B
17	1	131	18:36	18:50	B
18	1	45	18:37	18:50	A
19	2	132	18:37	18:50	A
20	6	132	18:37	18:50	B
21	2	45	18:40	18:50	A
22	5	44	18:40	18:50	A
23	3	130	18:40	18:51	B
24	6	131	18:40	18:51	A
25	2	43	18:43	18:53	A
26	2	45	18:43	18:53	A
27	3	130	18:45	19:15	B
28	1	131	18:45	19:15	B
29	1	44	18:50	19:02	A
30	2	43	18:55	19:15	A
31	6	131	18:55	19:15	A
32	3	44	18:55	19:08	A
33	2	45	18:55	19:08	A
34	1	42a	18:59	19:11	A
35	2	43	19:03	19:15	A
36	35a	133	19:03	19:15	B
37	1	43	19:03	19:15	A
38	3	129	19:04	19:18	B

Table 8.14: Characteristics of M_L, instance for location Eindhoven.

	$\tau_{\textbf{platform}}$	$\tau_{\textbf{park}}$	r_m	d_m	**side**
1	2	45	18:16	18:26	A
2	1	129	18:31	18:44	B
3	6	130	18:33	18:47	B
4	2	43	18:46	18:56	A
5	2	45	19:16	19:26	A
6	2	42a	19:46	19:56	A
7	3	131	19:56	20:10	B
8	2	45	20:16	20:26	A
9	3	130	20:26	20:41	B
10	2	43	20:46	20:56	A
11	3	130	20:56	21:19	B
12	1	44	21:01	21:14	A
13	2	43	21:16	21:26	A
14	3	44	21:26	21:40	A
15	1	42a	21:41	21:54	A
16	2	43	21:46	21:56	A
17	3	129	21:56	22:10	B
18	2	45	22:16	22:26	A
19	6	132	22:47	22:57	B
20	1	45	22:46	22:57	A
21	6	132	23:17	23:39	B
22	2	45	23:16	23:26	A
23	1	131	23:26	23:40	B
24	6	131	23:45	23:56	A
25	6	131	23:55	00:10	A
26	2	43	23:46	23:56	A
27	5	133	00:01	00:15	B
28	5	133	00:10	00:24	B
29	3	43	00:16	00:49	A
30	5	44	00:25	00:48	A
31	1	131	00:31	00:41	B
32	2	45	00:43	00:56	A
33	35a	133	01:02	02:04	B
34	3	133	00:56	01:13	B
35	35a	133	01:02	01:12	B
36	1	43	01:01	01:14	A
37	5	130	01:01	01:16	A
38	5	130	01:11	01:21	A
39	4	132	01:13	01:32	A
40	4	132	01:19	01:38	A
41	3	132	20:34	20:50	A
42	3	132	20:42	21:02	A
43	2	132	01:09	01:36	A
44	1	15	01:13	01:21	A
45	1	15	01:21	01:28	A

Table 8.15: Characteristics of M_O, instance for location Enschede.

i	$\tau_{\textbf{start}}$	$\delta_{\textbf{start}}$	$\tau_{\textbf{finish}}$	$\delta_{\textbf{finish}}$	r_m	d_m
1	414		404a	B	6:08	6:28
1	401	A	414		22:19	23:10
2	415		403	B	18:04	18:24
3	401	A	405b	B	18:19	19:10
3	405b	A	401	B	6:25	7:29
4	404a	A	415		18:31	18:42
4	415		404a	B	18:45	18:58
5	402	A	405b	B	18:49	19:40
5	405b	A	402	B	0:49	5:54
6	401	A	406		19:19	19:49
6	406		402	B	5:55	6:59
7	403		405b	B	19:36	20:06
7	405b	A	401	B	23:19	0:10
8	401	A	407	B	20:19	21:10
8	407	A	403	B	4:00	5:00
9	402	A	407	B	20:49	21:40
9	407	A	401	B	4:13	5:13
10	401	A	408	B	21:19	22:10
10	408	A	401	B	1:37	5:29
11	402	A	407	B	22:49	23:40
11	407	A	403	B	4:11	5:11
12	402	A	415		23:49	0:40
12	415	A	402		7:49	8:00
13	404a	A	415		0:01	0:23
13	415		404a	B	6:38	6:58

Table 8.16: Characteristics of M_P, instance for location Eindhoven.

i	$\tau_{\textbf{start}}$	$\tau_{\textbf{finish}}$	r_m	d_m
1	6	133	18:30	18:45
1	133	64	19:00	19:15
1	64	1	20:00	20:15
2	6	132	18:45	19:00
2	132	64	19:15	19:30
2	64	1	20:00	20:15
3	6	131	19:00	19:15
3	131	64	19:45	20:00
3	64	1	20:00	20:15

Table 8.17: Characteristics of M_Q, instance for location Eindhoven.

i	$\tau_{\textbf{start}}$	$\tau_{\textbf{finish}}$	r_m	d_m
1	1	133	18:30	18:45
1	133	3	19:00	19:15
2	2	132	18:30	18:45
2	132	4	19:00	19:15
3	3	131	18:30	18:45
3	131	5	19:00	19:15
4	4	133	18:30	18:45
4	133	6	19:00	19:15
5	5	132	18:30	18:45
5	132	1	19:00	19:15
6	6	131	18:30	18:45
6	131	2	19:00	19:15
7	1	133	18:35	18:50
7	133	3	19:05	19:20
8	2	132	18:35	18:50
8	132	4	19:05	19:20
9	3	131	18:35	18:50
9	131	5	19:05	19:20
10	4	133	18:35	18:50
10	133	6	19:05	19:20
11	5	132	18:35	18:50
11	132	1	19:05	19:20
12	6	131	18:35	18:50
12	131	2	19:05	19:20

Table 8.18: Characteristics of M_R, instance for location Eindhoven.

i	$\tau_{\textbf{start}}$	$\tau_{\textbf{finish}}$	r_m	d_m
1	2	45	18:16	18:34
1	45	5	18:36	18:49
2	1	129	18:31	18:46
2	129	2	5:57	6:28
3	6	130	18:33	18:47
3	130	2	4:18	5:18
4	2	43	18:46	18:56
4	43	5	19:04	19:19
5	2	45	19:16	19:32
5	45	5	19:34	19:43
6	2	42a	19:46	20:02
6	42a	5	6:04	6:16
7	3	131	19:56	20:10
7	131	3	6:05	6:47
8	2	45	20:16	20:33
8	45	5	20:34	20:50
9	3	130	20:28	20:49
9	130	4	4:00	5:00
10	2	43	20:46	20:57
10	43	5	23:04	23:17
11	1	44	21:01	21:14
11	44	1	7:41	7:42
12	2	43	21:16	21:32
12	43	5	22:34	22:48
13	3	44	21:26	21:43
13	44	16	5:20	6:04
14	1	42a	21:41	21:54
14	42a	3	4:16	5:16
15	2	43	21:46	22:00
15	43	3	22:07	22:19
16	3	129	21:56	22:10
16	129	16	4:24	5:24
17	2	45	22:16	22:36
17	45	5	7:04	7:19
18	6	132	22:47	23:01
18	132	5	6:30	7:11
19	1	45	22:46	23:00
19	45	4	5:19	5:39
20	6	132	23:17	23:39
20	132	1	5:56	6:56
21	2	45	23:16	23:30
21	45	5	23:31	23:49
22	1	131	23:28	23:42
22	131	2	6:23	6:28
23	6	12	23:45	23:59
23	12	5	5:11	5:45
24	2	43	23:46	0:00
24	43	3	5:57	6:39
33	5	133	0:01	0:15
25	5	44	0:25	0:48
25	44	1	5:02	6:02
26	1	131	0:31	0:47
26	131	2	4:57	5:57
27	133	44	0:53	1:53
27	44	5	4:27	5:27
28	1	43	1:01	1:17
28	43	1	6:13	6:28
29	5	130	1:11	1:25
29	130	2	6:26	6:56
5	15	1	4:51	5:51