Lagrangian Relaxation Methods for the Train Unit Shunting Problem

Thomas Verwaal

TU Delft

TECHNISCHE UNIVERSITEIT DELFT

MASTER OF SCIENCE THESIS IN COMPUTER SCIENCE

Lagrangian Relaxation Methods for the Train Unit Shunting Problem

Thomas VERWAAL

Supervisors:
Prof. Dr. M.M. de Weerdt
P.A.T.M. de Groot MSc ir.
I.K. Hanou

29th May 2026

Delft University of Technology

Lagrangian Relaxation methods for the Train Unit Shunting Problem

Master's Thesis in Computer Science

Algorithmics group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Thomas Verwaal

29th May 2026

Author
Thomas Verwaal

Title
Lagrangian Relaxation methods for the Train Unit Shunting Problem

MSc presentation
12-06-2026

Graduation Committee

Prof. Dr. M.M. de Weerdt	Delft University of Technology
Dr. G. Iosifidis	Delft University of Technology
P.A.T.M. de Groot MSc	Delft University of Technology
ir. I.K. Hanou	Delft University of Technology

Abstract

Rail transport is one of the most used forms of public transport, and apart from the timetable, effective shunting operations are an important part of operational efficiency and robustness. Trains not used in the timetable are parked at shunting yards, where shunting operations take place, which take 10-50% of trains' total transit time. Efficient planning of these operations is challenging due to relatively small yards that cannot be expanded easily, since they are located in urban areas. The management of trains outside of the timetable is an NP-hard problem known as the Train Unit Shunting Problem (TUSP). The TUSP concerns the matching, routing, and parking of arriving and departing trains. The current state-of-the-art Local Search (LS) approach is non-deterministic and struggles with the routing aspect of the TUSP. In this thesis, the TUSP is approached from a routing perspective with a deterministic algorithm. We apply Lagrangian Relaxation (LR) methods to the TUSP such that the problem can be split into per-train shortest path problems. Standard LR struggles with solving the TUSP due to symmetry in the trains and in the shunting yards. To address the symmetry, the approach is extended with Augmented Lagrangian Relaxation (ALR) and solved with the Alternating Direction Method of Multipliers (ADMM). Experimental results show that ADMM can be used to solve a somewhat simplified version of the TUSP and outperforms the LS approach on scenarios in which the arrival and departure of trains are more time restricted, i.e., smaller time windows. On larger time windows and for more trains, the LS outperforms the ADMM approach.

Preface

This thesis concludes my master's degree in Computer Science at Delft University of Technology. During my studies, I realized that I enjoy solving complex problems through logical reasoning, as well as analyzing solutions and identifying their limitations. This naturally led me to the algorithmics track, which helped me expand my knowledge of computational problem solving. In this thesis, I combine my interest in algorithmics with the practical challenges of the Train Unit Shunting Problem

I want to thank Issa Hanou and Peter de Groot for their weekly feedback, support, and discussions throughout this project. I would also like to thank Mathijs de Weerdt for his supervision and guidance during my thesis. Finally, I want to thank Leon Planken for his support in working with the Robust-Rail software, as well as the researchers at Sintef for their insights and discussions on the Train Unit Shunting Problem.

Thomas Verwaal

Delft, The Netherlands
29th May 2026

Contents

Chapter 1

Introduction

One of the most used forms of public transport is rail transport, as in the Netherlands, where 10.1% of the distance traveled was rail transport in 2023 [CBS]. The Dutch Railways (NS), the main train operator in the Netherlands, had over one million train journeys daily in 2024 [NS]. To facilitate these journeys, a large number of trains are needed. The efficient management of these trains is important for both reducing delays and reducing operational costs. A major part in this is the management of trains outside of rush hours and at night, when a large part of the trains are not used. These trains not participating in the timetable are moved to a shunting yard, where they can be parked and serviced, which includes tasks like cleaning or small maintenance. The movement and servicing may take 10–50% of trains' total transit time [Bontekoning and Priemus, 2004] and thus have a significant impact on the operational costs of trains.

Planning these operations in an efficient and feasible manner is thus important. In the Netherlands, shunting yards are relatively small compared to the number of trains that need to be managed and cannot be expanded easily since they are located in urban areas. Furthermore, over the years, the number of trains of the NS has increased significantly due to increased demand. The management of trains outside of the timetable has thus become increasingly difficult, and the efficient use of the available space more important.

At a shunting yard, the management of trains entails the matching, routing, and parking of arriving and departing trains. Arriving trains are a set of trains that have different types and are composed of different units. Departing trains consist of destination, train composition, and departure time. The matching problem concerns assigning arriving train units to positions in departing trains such that the composition of each departing train matches its required type and length. For example, an arriving train that consists of two units might have to be split if there are two departing trains that require those units. The routing and parking concern the moving of arriving trains to a parking track and of moving departing trains to

their destination. The problem of matching, routing, and parking of train units at a shunting yard is known as the *Train Unit Shunting Problem* (TUSP) [Freling et al., 2005]. The objective is to find a feasible plan such that all departures are executed as planned and no collisions occur. The soft objectives are to minimize train movement and group the movements of a train as much as possible to avoid excessive driver waiting time.

The NS currently uses a Local Search (LS) algorithm to find feasible solutions for the TUSP [van den Broek et al., 2022]. This is the first approach capable of solving the TUSP for real-world instances within a realistic time. In this thesis, we consider five minutes as a realistic time for use in practical planning operations, as this allows for a quick response in case of a change or disruption in the planning. However, the LS method is non-deterministic and struggles with the routing aspect of the TUSP. In this thesis, we propose to apply Lagrangian Relaxation (LR) methods [Fisher, 1981] such that the TUSP can be split into per-train shortest path problems. By modeling the TUSP as shortest path problems, this approach directly targets the routing component of the problem. These shortest path problems can then be solved efficiently with shortest path algorithms, as shunting yards are relatively small. Furthermore, LR methods are deterministic, in contrast to the LS approach, and can thus provide repeatable and consistent performance. A deterministic approach also adds stability in the planning process.

The LS approach solves an extension of the TUSP in which service task scheduling is included. Servicing tasks are tasks such as cleaning and maintenance. These tasks can only be performed at specific locations in a shunting yard. The extension of the TUSP with service tasks is known as the *Train Unit Shunting Problem with Service Scheduling*(TUSPwSS)[van den Broek et al., 2022]. For simplicity and since the main complexity of the problem comes from the interaction between the matching, routing, and parking [van den Broek et al., 2022, Kamenga, 2020], we do not consider servicing tasks as part of the problem in this research and focus on the TUSP. As a result, the LS approach addresses a more general and complex problem than the one considered in this thesis

In this thesis, we study a simplified version of the TUSP; the exact relaxations are discussed in Chapter 4. We model the TUSP as a Mixed Integer Linear Program (MILP) and apply standard LR. To address symmetry issues that arise when applying standard LR, we extend our approach with Augmented Lagrangian Relaxation (ALR), which we solve with the Alternating Direction Method of Multipliers (ADMM) [Boyd et al., 2011]. In this thesis, we aim to answer the following research question:

- **RQ**: How well can LR methods be applied to a simplified version of the TUSP?

To answer this question, we formulate the following sub-questions:

SQ1 How can the routing, parking, and matching components of the TUSP be modeled for LR methods?

SQ2 Can LR methods find feasible solutions for TUSP scenarios?

SQ3 What are the main limitations of LR methods for the TUSP?

SQ4 How does the performance of ADMM compare to the LS algorithm, in terms of the number of feasible solutions found and computation time?

SQ5 What is the impact of the time window on the performance of ADMM compared to the LS algorithm?

In this thesis, we show how shunting yards can be modeled as a time-extended graph and show a model that allows the TUSP to be split into per-train shortest path problems with LR methods. As the standard LR method struggles with symmetry, we extend it with ALR, which we solve with ADMM. Our experimental evaluation shows that ADMM can be used to solve the TUSP and performs better than the LS algorithm on scenarios with a smaller time window, while it struggles with scenarios that have a larger time window and more trains.

The main contribution of this thesis is a deterministic ADMM approach that can solve a simplified version of the TUSP. This approach has similar performance to the state-of-the-art non-deterministic LS algorithm.

Chapter 2

Background

In this chapter, we aim to give the reader sufficient background information to understand the methods we apply in this thesis to the Train Unit Shunting Problem (TUSP). We start by explaining Mixed Integer Linear Programming (MILP) and how Lagrangian Relaxation (LR) can be applied to MILP problems. After which, we explain Augmented Lagrangian Relaxation (ALR) and the Alternating Direction Method of Multipliers (ADMM). We then describe the current state-of-the-art Local Search (LS) algorithm for the TUSP, which serves as the baseline for the experiments.

2.1 Mixed Integer Linear Programming & Lagrangian Relaxation

MILP is a form of mathematical programming; an MILP model consists of a linear objective, a set of linear constraints, and a set of decision variables, of which at least one has to be integer [Nemhauser and Wolsey, 1988]. If a linear mathematical programming model has no integer decision variables, it is a Linear Programming (LP) model. LP problems are polynomially solvable [Khachiyan, 1979], whereas MILP problems are generally NP-hard. In practice, MILP models with fewer integer variables are generally easier to solve. The goal in MILP is often to minimize (or maximize) the objective function; in some cases, the objective is omitted, and the goal is to determine whether a feasible solution exists. As an example, we define an MILP:

$$Z = \min_{x,y}\{2x + y : x + y \geq 3, x, y \in \{0, 1, 2\}\} \tag{2.1}$$

Here, the goal is to find an x and y value such that the objective $2x + y$ is minimized without violating the constraint $x + y \geq 3$. An MILP problem is often solved using the model-and-solve paradigm, where the problem is formulated as a mathematical model and then solved by a general-purpose solver. The advantage of this paradigm is that it splits the modeling and solving, meaning that a new algorithmic

approach does not need to be developed for every problem. This is why substantial research effort has been invested in the development of MILP solvers [Bixby, 2012]. These solvers apply techniques like Branch-and-Cut and decomposition methods [Wolsey, 1998]. The solvers often apply LP relaxations to provide lower bounds for Branch-and-Cut algorithms. Even though MILP solvers are highly optimized, they, in general, do not scale well and struggle to solve specific problems. An alternative to solvers is using techniques, such as heuristics, or LR [Fisher, 1981].

2.2 Lagrangian Relaxation (LR)

In LR, certain hard constraints of a problem are relaxed into the objective function by introducing Lagrangian multipliers. Often, the constraints are relaxed that couple variables, such that the relaxed problem can be split into independent subproblems. These constraints are then no longer hard, and violations of these constraints are penalized through the objective function instead. If such a constraint is satisfied, it is rewarded through the objective function. The goal in LR is to find Lagrangian multiplier values that lead to good solutions. For the example in Equation 2.1, the relaxation of the constraint results in the following relaxed problem with multiplier λ:

$$L_\lambda = \min_{x,y}\{2x + y + \lambda(3 - x - y) : x, y \in \{0, 1, 2\}\}. \tag{2.2}$$

LR is often applied to problems in which relaxation of a constraint allows the problem to be separated into subproblems [Fisher, 1981]. This is also the case for our example, where Equation 2.2 can be rewritten into:

$$L_\lambda = \min_{x,y}\{(2 - \lambda)x + (1 - \lambda)y + 3\lambda : x, y \in \{0, 1, 2\}\}. \tag{2.3}$$

Which can then be split into two separate subproblems, where we leave out the constant 3λ:

$$L_\lambda = \min_{x}\{(2 - \lambda)x : x \in \{0, 1, 2\}\}, \quad L_\lambda = \min_{y}\{(1 - \lambda)y : y \in \{0, 1, 2\}\}. \tag{2.4}$$

These two subproblems are coupled through the Lagrangian multiplier λ. In LR, the subproblems are solved separately, after which the problems are solved again with an updated Lagrangian multiplier. There are several ways to update the Lagrangian multiplier; the most common is the subgradient method, which is the method we use in this thesis. For our example, the subgradient method is Equation 2.5, where k is the iteration. When using the subgradient method, the penalty increases if a constraint is violated, and decreases if a constraint is satisfied.

$$\lambda_{k+1} = \max(0, \lambda_k + \frac{1}{k}(3 - (x + y))) \tag{2.5}$$

The computation in LR is generally stopped when the Lagrangian multiplier or the Lagrangian value L stabilizes. By splitting a problem into multiple subproblems, LR allows for parallel computation of the subproblems, which can lead to a significant reduction in computation time. In Table 2.1 we show how LR solves our example problem. Here, k is the iteration, λ is the Lagrangian multiplier value, x and y are the values of x and y that are the optimal solutions to the separate subproblems in Equation 2.4, and L is the Lagrangian objective value as in Equation 2.2.

k	λ	x	y	L
0	0.00	0	0	0.00
1	3.00	2	2	3.00
2	2.50	2	2	3.50
3	2.17	2	2	3.83
4	1.92	0	2	3.92
5	2.12	2	2	3.88
6	1.95	0	2	3.95
7	2.09	2	2	3.91

Table 2.1: Iterations of the Lagrangian Relaxation, showing the solutions of the subproblems and the corresponding Lagrangian value.

In Table 2.1, we see L converge to 4, which is the optimal solution value of Z. λ converges to 2 which leads to the values of x and y oscillating between $(0, 2)$ and $(2, 2)$ in future iterations. For our example, LR finds one feasible solution, namely $x = 2$ and $y = 2$; this is not the optimal solution, which is $x = 1$ and $y = 2$.

It is often the case that LR is not able to find the optimal or even a feasible solution. In these cases, the Lagrangian value found by LR, which is guaranteed to be a lower bound of the original problem, is often used as such in other algorithms. In some cases, if no feasible solution is found, the solution found by LR is made feasible by repairing conflicts.

Formally, LR solves the Lagrangian dual problem, in which the goal is to maximize the Lagrangian value over all non-negative multiplier values. For a fixed multiplier λ, the Lagrangian relaxation provides a lower bound on the optimal value of the original problem. The objective is therefore to find multiplier values that maximize this bound [Hooker, 2024].

2.3 Augmented Lagrangian Relaxation (ALR) & The Alternating Direction Method of Multipliers (ADMM)

While LR can be used to separate the subproblems and can sometimes find solutions, it has slow convergence and oscillatory behavior. ALR can improve the convergence of LR [Boyd et al., 2011]. It adds a quadratic penalty term of the relaxed constraints to the objective, which penalizes constraint violations further, leading to better convergence. ALR, however, destroys the separability of LR since the variables are now coupled through the quadratic term. For our example, ALR leads to the objective in Equation 2.6, where ρ is the penalty parameter.

$$L_{\lambda,\rho} = \min_{x,y}\{2x+y+\lambda(3-x-y)+\max(0, \frac{\rho}{2}(3-x-y)^2) : x, y \in \{0,1,2\}\} \quad (2.6)$$

ADMM [Boyd et al., 2011] is often used to recover separability in ALR by alternating the optimization of the different variables. In ADMM, the problem is split into a sequence of interdependent subproblems by block coordinate descent and linearization techniques [Song and Cheng, 2022]. The solution for one variable is solved with the values of the other variables fixed:

$$x^{k+1} = arg\min_{x} 2x + \lambda^k(3 - x - y^k) + \max(0, \frac{\rho}{2}(3 - x - y^k)^2) \quad (2.7)$$

$$y^{k+1} = arg\min_{y} y + \lambda^k(3 - x^{k+1} - y) + \max(0, \frac{\rho}{2}(3 - x^{k+1} - y)^2). \quad (2.8)$$

ADMM thus takes the values of the other variables into account when solving for one variable, whereas in LR, the variables are only coupled through the Lagrangian multiplier. Although ADMM allows for separability, one of its downsides is that it may fail to converge and can have cyclic behavior, which is generally not the case when solving ALR without ADMM. In Table 2.2, we show how ALR with ADMM solves our example problem, with $\rho = 1$. We see that ADMM finds the optimal solution and converges faster than LR. The main drawback of ALR and ADMM is that the subproblems can no longer be solved in parallel.

k	λ	x	y	L
0	0.00	1	1	3.50
1	1.00	1	2	3.50
2	1.00	1	2	4.00
3	1.00	1	2	4.00

Table 2.2: Iterations of ADMM, showing the alternating solutions (x, y), and the corresponding augmented Lagrangian value L_ρ.

2.4 Local Search (LS) for the TUSPwSS

The current state-of-the-art algorithm for the TUSP extended with service task (TUSPwSS) is an LS approach developed by [van den Broek et al., 2022]. In this section, we first give a short description of the TUSPwSS and then explain what LS is and how the TUSPwSS was transformed into an optimization problem. After this, we explain how an initial solution is created and describe the four search neighborhoods used to modify it.

The TUSPwSS concerns the matching, routing, parking, and servicing of train units at a shunting yard. A shunting yard consists of a set of tracks where trains can be parked, which are connected by switches. In a TUSPwSS instance, an arriving and departing schedule of trains is given, in which the arrival and departing times, locations, and compositions of trains are given. Each train consists of train units that have varying lengths and types. For each train unit, it is also defined what servicing tasks have to be performed. Certain servicing tasks can only be performed at certain tracks. The goal is to assign each train unit to a position in a departing train with matching type, scheduling its required servicing, and determining feasible routes through the yard while avoiding collisions. Ideally, the total movements are minimized.

LS is a heuristic method for solving optimization problems. It starts with an initial solution to a problem, and then iteratively improves the solution by selecting a neighboring solution. van den Broek et al. [2022] used simulated annealing [Kirkpatrick et al., 1983, Černỳ, 1985] to select a neighboring solution for the next iteration. In simulated annealing, a candidate solution is generated by applying a randomized neighborhood move to the current solution. This solution is accepted as the solution for the next iteration if it is an improvement over the current solution. If it is not an improvement, it is accepted with a probability based on the difference in objective value between the candidate and the current solution, and based on the state of the search process. Worse solutions are accepted with a probability to avoid getting stuck in local optima.

[van den Broek et al., 2022] transformed the TUSPwSS into an optimization problem by relaxing the temporal and parking constraints and penalizing violations of these constraints in the objective function. This allows them to start with a solution to this relaxed problem, which can then be improved with LS until a feasible solution is found to the original problem. The temporal constraints ensure that train units enter the yard upon arrival and depart on time. The parking constraints ensure that train units are parked without exceeding the track capacity or blocking train unit movements. Both LS and LR relax constraints. In LS, this is done to allow the algorithm to explore the solution space more freely, while in LR, constraints are relaxed to simplify the problem and allow decomposition.

An initial solution for the relaxed problem is constructed with a sequential algorithm. First, a matching between the incoming and outgoing train units, which form a bipartite graph, is constructed by using the Hopcroft-Karp algorithm [Hopcroft and Karp, 1973]. Based on this matching, the number of required splits needed to transform the incoming train units into the desired departure compositions is computed. A short, simple simulated annealing algorithm is used to reduce the number of splits. Next, the service schedule is constructed by a list-scheduling strategy. Tasks are added to the schedule of a corresponding resource in order of increasing due date and based on resource workload. The movements to and from the service locations are computed next. Finally, the parking locations of the train units are assigned randomly without taking the track occupation into account. The initial solutions created with this algorithm have many violations for problems of significant size.

2.4.1 Search Neighborhoods

The neighboring solutions of the initial solution are solutions where certain aspects of the planning are altered. In [van den Broek et al., 2022], four search neighborhoods are defined such that the different aspects of the TUSPwSS are considered.

The first neighborhood is a change in parking location, which can solve conflicts in track capacity and collisions. This neighborhood contains all shunting plans that can be constructed by changing the parking location of a train. The parking location of a train unit can be changed by selecting two consecutive movements, m_1 and m_2, and assigning the destination of m_1 and the origin of m_2 to a different track.

The second neighborhood is a change in the routing. This neighborhood consists of three sub-neighborhoods. First, the shift movement neighborhood, in which a movement is scheduled to be earlier or later in the movement order. Second, the insert movement neighborhood, where two extra movements are inserted to move a train to a different track temporarily. Both these neighborhoods can resolve collisions: the shift movement neighborhood by moving a train unit earlier, out of the way of another train unit, or by letting a train arrive later, and the insert movement neighborhood by moving a train out of the way temporarily. The third sub-neighborhood is the remove movement neighborhood, in which a redundant train movement is removed, leading to fewer movements. A train movement is redundant if it can be removed without creating new conflicts.

The third neighborhood is a change in the servicing schedule. This neighborhood contains all valid solutions, in which the order of two consecutive service tasks is swapped, that use the same resource, or involve the same train. It also contains the solutions obtained by assigning a service task to a different resource.

The final neighborhood is the matching neighborhood, in which the assigned departure of the two trains is swapped. Neighboring solutions are selected randomly. If a solution is changed in one dimension, then the plan is also changed accordingly in the other dimensions, such that the solution remains valid.

In LS, the initial solution is changed by repeatedly selecting better or worse neighboring solutions. The chance that a worse solution is selected is reduced over time, and the LS runs until a feasible solution is found or until no more improved solutions can be found and worse solutions are no longer accepted.

The LS approach is a solid approach for solving the TUSP, as it is the first algorithm that is capable of solving the complete problem for real-world instances in a realistic time for use in practical planning operations. The main drawbacks of the LS approach are that it is non-deterministic, and thus does not consistently find solutions, and it can get stuck in local optima. Furthermore, the author mentions that the routing appears to be a bottleneck in many instances.

Chapter 3

Related Work

In this chapter, we give an overview of previous research on the Train Unit Shunting Problem (TUSP) and discuss relevant techniques from the literature that could be used to solve the TUSP. To provide an overview of previous approaches that were applied to the TUSP, we start by discussing the most relevant works done on the TUSP and the techniques used. We then describe work done on the TUSP without matching and (de)coupling at Chinese railways, where Lagrangian Relaxation (LR) methods were used. Afterwards, we discuss relevant research on Multi-agent Path Finding (MAPF) and LR. We conclude by outlining why LR methods could be well-suited for the TUSP.

3.1 Train Unit Shunting Problem

Freling et al. [2005] first introduced the TUSP, which they proved is NP-hard. They considered the matching and parking subproblems, which they solved sequentially. First, they used CPLEX to solve the mathematical models of the matching subproblem, and second, a column generation heuristic was used to assign the matched train units to tracks. The main drawbacks of this approach are that it is possible that the solution for the matching leads to an infeasible problem for the parking, and that the routing subproblem is not considered.

Lentink et al. [2003] extended the TUSP by including the routing subproblem. They used the algorithm proposed by Freling et al. [2005] (published later) to solve the matching subproblem and also solved the subproblems sequentially. The routing is solved last, and it is possible that no feasible routing solution exists, given the solution for the matching and parking. In this case, the solutions for the matching and parking have to be recomputed with an extra constraint.

Instead of solving the matching and parking sequentially, Kroon et al. [2008] solved them simultaneously, instead using CPLEX. Their Mixed Integer Linear Programming (MILP) approach leads to a large model that takes several hours to

find a feasible solution, and the routing is not even considered yet.

In a more recent study by Haahr et al. [2017], three novel approaches, a Constraint Programming formulation, a Column Generation approach, and a Greedy Construction Heuristic were compared to two existing methods, an MILP [Kroon et al., 2008] and a Two-stage Heuristic [Freling et al., 2005]. Their main conclusion was that the different approaches all have different strengths and weaknesses depending on instance characteristics. This research, however, did not consider the routing.

Kamenga [2020] proposed four algorithms, each solving the different subproblems in a different order, partially sequential and partially integrated. They concluded that the performance of these algorithms does not differ much, but that different instance characteristics may imply different relative performance. Furthermore, they concluded that the parking subproblem is the subproblem that complicates the TUSP the most, and that the servicing subproblem only has a minor impact on the difficulty of the problem. In all four algorithms, the routing is solved last. Their algorithm required twenty minutes to find a solution in the worst case. They remark that the number of paths per train strongly affects the size of the parking and routing subproblem.

Most of the above approaches used MILP to solve the TUSP. MILP is a cornerstone in operations research and can be used for a variety of problems [Clautiaux and Ljubić, 2025]. The advantage of using MILP to solve a problem is that many available solvers can solve many problems in a reasonable time. Furthermore, the solutions provided by MILP are provably optimal. However, solving NP-hard problems exactly leads to exponential growth of solution time for larger problem instances. As shown by the discussed research, MILP is not able to solve instances of realistic sizes for the TUSP in a realistic time for use in practical planning operations.

The current state-of-the-art solution is the first algorithm capable of solving the complete problem for real-world instances in a realistic time. van den Broek et al. [2022] developed a Local Search (LS) algorithm to solve the TUSP, extended to include service task (TUSPwSS). The LS approach solves the TUSP as an integrated optimization problem rather than solving the subproblems sequentially. This algorithm can solve instances for different shunting yards within a few minutes. It can solve harder instances than a decision support tool developed by NS, and outperforms current algorithms for the TUSP without service tasks. They also demonstrated that the flexibility to move a train to a different track during parking (reallocation) is essential to finding feasible plans. The main drawbacks of the LS approach are that it is non-deterministic, and thus does not consistently find solutions, and it can get stuck in local optima. Furthermore, the author mentions that the routing appears to be a bottleneck in many instances.

Another approach to solve the TUSP was by Athmer [2021], since the LS approach is quite dependent on the initial solution, they used an evolutionary algorithm to solve the TUSP, which is less dependent on the initial solution. Their approach was able to outperform the LS approach at one location but not at another.

3.1.1 TUSP without Matching and (De)coupling

At Chinese railways, a problem slightly simpler than the TUSP occurs. The difference with the TUSP is that they do not have to consider the (de)coupling of trains and the matching subproblem, and they only have two train types: long trains of sixteen train units and short trains of eight train units.

In recent studies, Wang et al. [2022] used Integer Linear Programming to model the problem and solved it using Gurobi; they were able to find solutions in 30 minutes. In another study by Xu and Dessouky [2022], the problem is modeled as a minimum-cost multi-commodity network flow problem and solved using Lagrangian relaxation. Their approach required more than one hour to find a solution for about 20 trains. An extension of this work was done by Shi et al. [2023], where they used Augmented Lagrangian Relaxation (ALR) and the Alternating Direction Method of Multipliers (ADMM) to solve symmetry problems in LR. In this approach, they also considered the reallocation of trains during parking. Their approach was able to get very close to a feasible solution within one hour; they only tested their approach on one instance.

To our knowledge, these papers are the only research that applied Lagrangian relaxation to the TUSP. Their model, a two-layer space-time network, differs significantly from how we model the TUSP in this research, and we also include the matching subproblem, making our research stand apart. This research does show that ALR can be applied to a problem similar to the TUSP version considered in this thesis.

3.2 Multi-Agent Path Finding (MAPF)

In a MAPF problem, a set of agents is given that have a start and goal location in a graph or grid. The goal in MAPF is to find a path for each agent from its start to goal location while avoiding collisions between agents, ideally minimizing either the sum of costs of all paths or the total time needed. The MAPF problem is similar to the routing subproblem in the TUSP. In the context of MAPF for the TUSP, trains can be seen as agents, the shunting yard as a graph, train collisions as collision constraints, and the arrival and departure tracks as start and goal locations, respectively. Using MAPF algorithms to solve the routing for the TUSP is thus a logical choice.

The current state-of-the-art algorithm for MAPF is conflict-based search (CBS) [Sharon et al., 2015]. CBS is a search-tree algorithm that finds the optimal paths for all agents. van Cuilenborg [2020] applied CBS to the TUSP. In this research, they altered Conflict-Based Search (CBS) such that it can be applied to the TUSP, and applied different variations of CBS to the TUSP. Although they were able to apply CBS to the TUSP, their approach was slower than the LS approach and found fewer feasible solutions.

3.3 Lagrangian Relaxation

As discussed in Section 3.1, using MILP to solve NP-hard problems often leads to unrealistic solution times for problem instances of realistic size. LR [Fisher, 1981] is a method that can reduce the complexity of MILP problems by decomposing them into smaller subproblems [Bragin, 2023]. LR decomposes a problem into smaller subproblems by relaxing some hard constraints into the objective and introducing Lagrangian multipliers (penalty terms); these subproblems can often be solved in parallel. The relaxation loses guaranteed feasibility and optimality for the original problem since the solutions for the subproblems can be conflicting in the original problem. LR is often used as a heuristic, where relaxed solutions are used to guide primal recovery. Primal recovery is the process of converting solutions to the relaxed problem into feasible solutions for the original problem. There are a lot of different methods that can be used in LR, such as the subgradient method, augmented methods, column generation, and dual ascent methods [Guignard, 2003], making LR applicable to a large variety of problems.

Over the years, LR has been successfully applied to various problems such as the traveling salesman problem [Held and Karp, 1970], vehicle routing problems [Fisher and Jaikumar, 1981, Kohl and Madsen, 1997], and railway routing and scheduling [Zhou and Teng, 2016, Brännlund et al., 1998, Keaton, 1989]. These problems have routing and scheduling components similar to the TUSP, indicating that LR could be a useful tool for solving the TUSP. However, the TUSP has additional components that distinguish it from these problems. In particular, it combines routing with matching decisions and occurs at shunting yards with parallel tracks where trains can block each other, and often involves limited space and time constraints. Furthermore, there are often a few or even only one arrival and departure track. These components introduce additional conflicts and dependencies not present in standard routing or scheduling problems. Even with these components, the TUSP can be modeled as an MILP and is decomposable into multiple shortest path subproblems. This decomposition aligns well with the strengths of LR, and LR could thus be a powerful tool for solving the TUSP from a routing/MAPF perspective.

3.4 Conclusion

The research discussed on the TUSP has shown that exact methods are not able to solve the TUSP in a realistic time. Heuristic methods, LS, and evolutionary algorithms provide better performance. Although modeling the TUSP as a MAPF problem is a logical choice, previous research showed it was not able to perform better than the LS approach. LR is a decomposition approach that has been used on numerous NP-hard problems. Aside from a similar problem on the Chinese railways, LR methods have not been applied to the TUSP. Furthermore, in the discussed research on the TUSP, the routing was either not considered, solved last, or, in the case of van den Broek et al. [2022], a bottleneck in many instances. In this research, we aim to use LR methods to decompose the TUSP into multiple shortest path subproblems, focusing on the routing subproblem. The flexibility of reallocation, which is essential to finding feasible solutions, is also present in our model.

Chapter 4

Problem Statement

The train unit shunting problem (TUSP) concerns the matching, routing, and parking of train units at a shunting yard. The input of an instance of a TUSP is two-fold: a location and a scenario. The location describes the characteristics of the yard, and the scenario gives the arriving and departing trains and the total time window over which they arrive and depart. A valid solution to a TUSP instance is a solution without collisions and where each departure is reached, preferably on time.

The TUSP is a hard problem to solve, with many details that complicate the problem. Solving the routing, matching, and parking individually can already be difficult for a large number of trains, but solving these problems together is even more complex. A change in the matching can, for example, have an impact on how trains have to be routed and parked. The TUSP was shown by Freling et al. [2005] to be NP-hard.

In this chapter, we give a formal description of the TUSP based on the descriptions in Lentink [2006] and van den Broek [2016]. This problem has several important subproblems, namely the matching, routing, and parking. We first describe the input of the problem, the specifics of a yard, and a scenario of arriving and departing trains. We then discuss the details of these subproblems, starting with the matching of the arriving trains to the departing trains, followed by the routing and parking of the trains. The routing and parking are connected and thus described together. To illustrate the problem, this chapter ends with an example of both a shunting yard and a TUSP instance.

In practice, trains consist of multiple units that can be coupled and decoupled. In this thesis, each train consists of one unit, and we thus do not consider (de)coupling of train units. Including this was outside of the scope of this thesis and is a major simplification of the TUSP. Furthermore, we leave out servicing tasks and have some additional minor relaxations and simplifications. These additional simplifications do not affect the main complexity of the problem, which comes from the

interaction between the subproblems.

4.1 Location

This section describes the characteristics of a shunting yard, explains the different types of switches, and explains the saw movement. Each shunting yard has two sides, commonly referred to as the A-side and the B-side. A shunting yard consists of a set of tracks TR and every track tr has a length $\ell(tr)$, an A-side $as(tr)$, and a B-side $bs(tr)$. The $as(tr)$ and $bs(tr)$ of a track are the sides that are the closest to the A-side and B-side of the yard, respectively. The $as(tr)$ and $bs(tr)$ of a track are connected to either a bumper or a switch. A bumper is the end of a track, and a switch is a connection to another track. It is also possible to have a series of switches after one another that can be used to reach more tracks than with a single switch. Tracks with a switch on both sides are called free tracks, and tracks with a bumper on one side are called LIFO tracks. For every track, it is defined whether parking and/or reversals (saw movements) are allowed.

There are three types of switches: a regular switch, a crossover, and an English switch. The regular switch has two tracks connected on one side and one track on the other side. A crossover connects two parallel tracks, allowing trains to switch to the other track. An English switch is a crossing of two tracks on which trains can switch to the other track. An English switch is similar to the crossover, but two trains cannot traverse the switch at the same time, even when they remain on their own track. An example of these switches can be seen in Figure 4.1. Parking and saw movements are not allowed on switches. In this research, we assume that switches can be controlled remotely.

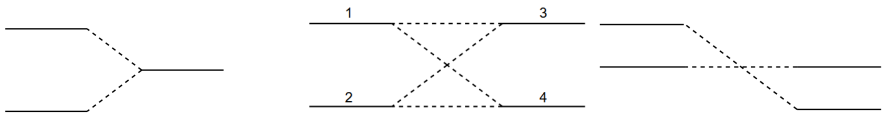


Figure 4.1: Regular switch (left), crossover (middle), and English switch (right). The solid lines represent tracks, and the dotted lines represent switches.

Each switch sw has an A-side $as(sw)$, and a B-side $bs(sw)$. We define $as(sw)$ and $bs(sw)$ to be an ordered list of tracks that the switch is connected to on the respective sides. Each track in $as(sw)$ is connected to each track in $bs(sw)$. The lists are ordered such that for a crossover, it is defined that a train can move from the first track in $as(sw)$ to the first track in $bs(sw)$ at the same time as another train moves from the second track in $as(sw)$ to the second track in $bs(sw)$. For the crossover in the figure, $as(sw) = [1, 2]$ and $bs(sw) = [3, 4]$. At this crossover, a train can move between tracks 1 and 3 at the same time as another train moves between tracks 2 and 4.

A saw movement is the operation of reversing the direction of a train unit. To perform a saw movement, the control of the train must be transferred to the tail of the train, and the driver must walk to the other end of the train to resume the movement. In general, saw moves are time-consuming operations.

A shunting yard is connected to the railway network by one or more gateway tracks. Trains can only arrive and depart from gateway tracks. For most locations, parking and reversal are not allowed on gateway tracks. In this research, we assume that the shunting yard is empty at the start and end of a scenario.

4.2 Scenario

In this section, we describe the characteristics of scenarios. A scenario consists of an arrival schedule, a departure schedule, and a time window $(0, ..., T_{\max})$ [Lonyuk, 2024]. From the arrival schedule, we get a set of trains (agents) A; every train a has an arrival time $at(a)$, an arrival track $ag(a)$, and a type $\theta(a)$. The departure schedule is a set of departures D, and each departure d has a departure time $dt(d)$, a departure track $dg(d)$, and a type $\theta(d)$. The departure time of some departures can be before the arrival time of arriving trains. Trains require a driver, but not a locomotive, to move and can move bi-directionally.

4.3 Matching

In this section, we define the matching subproblem. The matching problem concerns the matching of arriving trains to departing trains. There are many possible ways to match trains, because trains of the same type can be used interchangeably. The objective is to match each train $a \in A$ to a departure $d \in D$. A train can be matched to a departure if it arrives before the departure time $at(a) < dt(d)$, and if the train types match $\theta(a) = \theta(d)$.

4.4 Routing & Parking

This section describes how we model the shunting yard and how we use this model to formally define the routing and parking subproblems. The routing and parking concern the moving and waiting (parking) of trains at the shunting yard. Since trains cannot move past each other on a track, the order in which they are parked is important, especially when trains can only leave a track on one side.

To formally define the routing and parking, we model the shunting yard as a time-extended graph; this graph is based on the description by Mulderij et al. [2020]. This model is abstract and does not capture all practical details of the TUSP; it

is thus not a fully realistic model of the TUSP. In this definition, we assume all trains are 100 meters long and divide a track into multiple nodes by dividing its length by 100. A track of 420 meters is thus divided into four nodes with edges between them; each node can fit one train. Let $G = (TR,\ E)$ be a connected, undirected graph that consists of a set of tracks (nodes) TR and a set of edges $E \subseteq TR \times TR$. The edges originate from the switches. For each switch sw, an edge is added between each track in $as(sw)$ and each track in $bs(sw)$. We define C as the collection of conflicting edge sets, where each $c \in C$ consists of edges that cannot be used simultaneously. All edges that originate from the same regular switch or English switch are conflicting and belong to the same set c. For the crossover, only the edges that cross each other are conflicting. Two conflicting edge sets originate from a crossover. Edges originating from a series of switches are also conflicting, since parking in between switches is not allowed. The edge set also contains self-edges for tracks on which parking is allowed. Let $V = TR \times T$ be the time-extended node set, where each node $v \in V$ is a combination of a track and a time, $v = (tr,\ t)$. We define the time-extended directed edge set $\hat{E} \subset V \times V$, where an edge $e = (v_1,\ v_2) = ((tr_1,\ t_1), (tr_2,\ t_2)) \in \hat{E}$ if and only if $t_2 = t_1 + 1$ and $(tr_1,\ tr_2) \in E$. The reverse edge e' of e is $e' = ((tr_2,\ t_1), (tr_1,\ t_2))$. A reverse edge e' belongs to the same conflicting edge sets as its corresponding edge e. The directed graph $\hat{G} = (V, \hat{E})$ is then the space-time graph corresponding to G.

In the routing and parking subproblem, we assume that each train has been assigned to a departure through the matching and thus has a departure track $dg(a)$ and time $dt(a)$. The goal of the routing and parking problem is then to find a path for each train to its departure track such that there are no collisions. In terms of the time-extended graph, this means a path has to be found for each train $a \in A$, from $v = (ag(a),\ at(a))$ to $v = (dg(a),\ dt(a))$. A path for an agent is a vector of tracks $(tr_0, tr_1, ..., tr_{k-1})$, where $tr_0 = ag(a)$ and $tr_k = dg(a)$, and $((tr_t,\ t), (tr_{t+1},\ t+1)) \in \hat{E}$ for all $t \in 0, 1, ..., k$. For $t > k$, the train is no longer present at the yard. A path has to be found for each train $a \in A$ such that no two trains use the same node at the same time and no two conflicting edges are used at the same time. A train is parked if $tr_i = tr_{i+1}$ and is moving otherwise.

Ideally, the total movement costs are minimized, and departures are reached on time. The cost $c(p)$ of a path p is the sum of the traversed edges' cost. Furthermore, the aim is also to ensure that the movements of a train are grouped as much as possible to avoid excessive operator waiting time.

In practice, every train's movement time and cost depend on type, length, and switch specifics. In van den Broek [2016], a train movement is estimated to take one minute and to operate a switch 30 seconds. A reversal is estimated to take two to four minutes. In this research, we assume that traversing an edge takes one minute and reversals are instant. Since an edge can cross multiple switches, a movement in our model takes significantly less time than in practice. Furthermore,

we set the cost of each non-self edge to one.

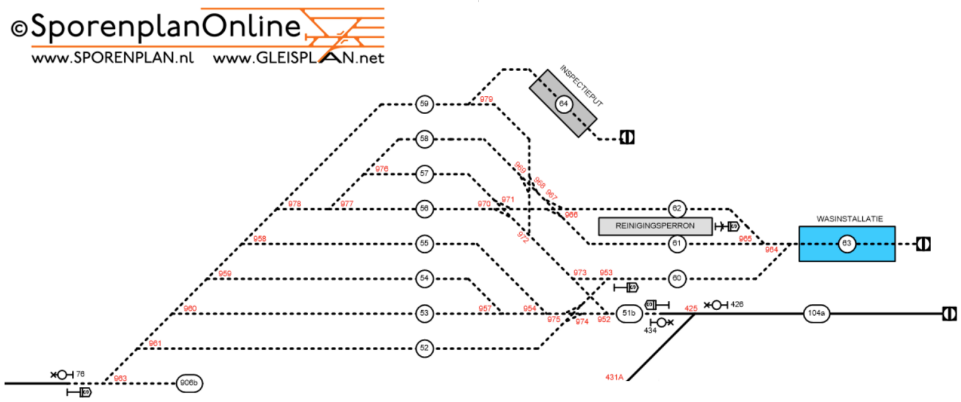


Figure 4.2: The Kleine Binckhorst shunting yard in Den Haag, gateway track at the bottom left.

4.5 Shunting Yard & TUSP Scenario Example

In this section, we begin by presenting an example of a shunting yard and how it is transformed into the graph described in Section 4.4. We then give an example of a scenario at this yard. Using this scenario, we illustrate that the matching, routing, and parking subproblems are interdependent.

The shunting yard on which we test our implementation in this thesis is the Kleine Binckhorst in Den Haag, depicted in Figure 4.2. The Kleine Binckhorst has only one gateway track (at the bottom left) and has both free tracks and LIFO tracks. In practice, shunting yards differ in layout and size. The distribution of free and LIFO tracks also varies, meaning yards contain significantly more or fewer cycles than others. For this example, we created a simple shunting yard based on the Kleine Binckhorst; this yard can be seen in Figure 4.3a. It has one gateway track, track 0, and two LIFO tracks, tracks 1 and 2. All tracks are 200 meters long, and there is one switch connecting the tracks.

The simple yard is transformed into our graph structure, which can be seen in Figure 4.3b. Track 0 is turned into node 0; it is not split into two nodes because parking is not allowed at the gateway track. Since we assume all trains are 100 meters long, tracks 1 and 2 are both split into two nodes, and an edge is added between these nodes. Switch 20 is turned into four edges: an edge from node 0 to nodes 1 and 2, and the reverse of these edges. For each node, aside from node 0, a self-edge is added to model parking. The edges $(0, 1), (1, 0), (0, 2)$ and $(2, 0)$ are conflicting and thus part of the same conflicting edge set.

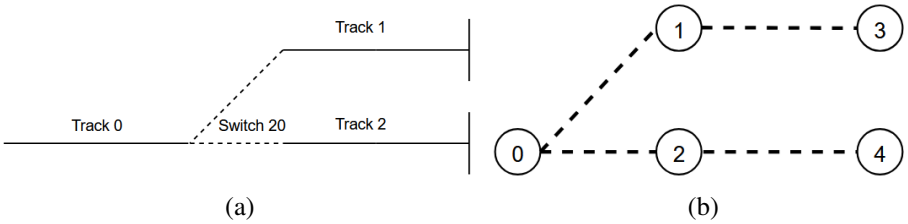


Figure 4.3: Simple yard example (a) and its graph representation (b).

The scenario we consider in this example has four arriving and departing trains; the exact schedule can be seen in Table 4.1. In this scenario, all trains arrive before a train departs. There is space for four trains in the yard, and it will thus be full after all trains arrive. Since each edge takes one timestep to traverse, and since there are only two timesteps between arrivals and departures, trains do not have time or space to reallocate between arrival and departure. Each train in this scenario can be matched to two departures.

Arrivals (id, $\theta(a)$, $ag(a)$, $at(a)$)	Departures ($\theta(d)$, $dg(a)$, $dt(a)$)
(1, SLT, 0, 0)	(SLT, 0, 9)
(2, SNG, 0, 2)	(SLT, 0, 11)
(3, SLT, 0, 4)	(SNG, 0, 14)
(4, SNG, 0, 6)	(SNG, 0, 16)

Table 4.1: Arriving and departing schedule for the example scenario.

To illustrate that the matching, routing, and parking are interdependent, we start by constructing a feasible matching. There are four feasible matchings for this schedule when only considering the matching subproblem. We consider two in this example and only focus on the matching of the SLT trains. In the first matching, we match the first arriving SLT train to the first departing SLT train and the second arriving SLT train to the second departing SLT train. For this matching, there is no feasible routing solution. The first SLT train has to be parked in the back since it arrives first and cannot be reallocated. It also has to depart first, which is impossible since its path to node 0 will be blocked by the other trains. For the second matching, we swap the matching of the SLT trains. This matching does have feasible routing solutions; the steps of one of these are listed in Table 4.2. When looking at the matching of the SNG trains and at the other feasible matching solutions, one will find that only one of the four feasible matching solutions has feasible routing solutions.

This example shows that the solutions to the matching, routing, and parking sub-

problems are interdependent. The impact of the time component of the TUSP is partially illustrated in this example. In a larger yard, the time required to traverse it can significantly affect whether matching, routing, and parking solutions are feasible within the available time.

Train ID	Start time	End time	Movement	Event
1	0	2	$0 \rightarrow 3$	Arrival
2	2	4	$0 \rightarrow 4$	Arrival
3	4	5	$0 \rightarrow 1$	Arrival
4	6	7	$0 \rightarrow 2$	Arrival
3	8	9	$1 \rightarrow 0$	Departure
1	10	11	$3 \rightarrow 0$	Departure
4	12	14	$2 \rightarrow 0$	Departure
2	14	16	$4 \rightarrow 0$	Departure

Table 4.2: Routing steps for each train.

Chapter 5

Methods

As discussed in Chapter 3, past research on the Train Unit Shunting Problem (TUSP) has shown that exact methods are not able to solve the TUSP in a reasonable time and that the current state-of-the-art solution is non-deterministic, where the planning of movements is a bottleneck. Lagrangian Relaxation (LR) is a heuristic method that is, in theory, deterministic. It allows the TUSP to be split into multiple shortest path sub-problems and thus approaches the TUSP from a routing/MAPF perspective.

In this chapter, we discuss the approach taken in this thesis to solving the TUSP version described in Chapter 4. We start by describing the Mixed Integer Linear Programming (MILP) model and how we apply LR to it. We then illustrate why a straightforward use of LR struggles with solving TUSP instances in which trains have to take detours, and the proposed solution to this problem. After this, we illustrate why the LR struggles with symmetry and a solution for these problems, which is extending the LR with Augmented Lagrangian Relaxation (ALR) and the Alternating Direction Method of Multipliers (ADMM). We then describe how the resulting per-train sub-problems can be solved with shortest path algorithms and end the chapter by giving pseudo-code of the ADMM approach. This chapter answers sub-question **SQ1** by showing how the routing, matching, and parking components of the TUSP can be modeled for LR methods and answers sub-question **SQ3** by discussing the symmetry problems of LR. An overview of the notations used in this chapter can be found in Table 5.1.

5.1 Mixed Integer Linear Programming Model

In this section, we describe a MILP model for the TUSP that we can apply LR to. This MILP model is based on the description of the TUSP in Chapter 4. We model the shunting yard as the time-extended graph described in Section 4.4, since a graph can model the layout of a yard. We use discrete time steps, where each minute is one timestep, and each edge takes one minute to traverse. A traversal

time of one minute for each edge is not fully realistic, as in practice, certain edges represent longer routes. Furthermore, edges between nodes originating from the same track also take one minute, whereas in practice, these movements take less time.

Notation	Description
TR	Set of tracks (nodes).
$E \subseteq TR \times TR$	Set of directed edges; includes a self-edge for every node if parking is allowed.
$c \subseteq E$	Set of conflicting edges.
$C := \{c_1, c_2, \ldots, c_m\}$	Set of conflicting edge sets.
$T_{\max}$	End of the time window; time is modeled in discrete steps.
A	Set of trains.
Θ	Set of train types.
$\theta(a) \in \Theta$	Type of train a.
$ag(a) \in TR$	Start location of train a.
$at(a) \in T$	Arrival time of train a.
$D \subseteq TR \times \Theta \times T$	Set of departures defined by departure track, train type, and time.
$D(\theta) := (i, \theta, t) \in D,\ \forall \theta \in \Theta$	Set of departures for trains of type θ.
$x_{a,i,j,t} \in \{0, 1\}$	1 if train a moves from node i at time t to node j, arriving at $t+1$.
$p_{a,i,t} \in \{0, 1\}$	1 if train a is located at node i at time t.
$y_{a,t} \in \{0, 1\}$	1 if train a is present in the yard at time t.

Table 5.1: Notations used in this chapter.

Previous research has already shown that finding a feasible solution to the TUSP is difficult, which is why the main goal is to find feasible instead of optimal solutions. A feasible solution is a solution without collisions and where each departure is reached on time. Although late arrivals at the destination are allowed in reality, our model does not allow for delays, since this was outside of the scope of this thesis. In practice, a feasible solution is determined by the scheduling of the train drivers. Whether or not a feasible schedule exists for the train drivers depends partially on the number of train movements in a solution to the TUSP. The objective in the model is thus to minimize the total train movements:

$$\min \text{obj}(x) := \sum_{t=0}^{T_{\max}-1} \sum_{\substack{(i,j) \in E \\ i<j}} \sum_{a \in A} (x_{a,i,j,t} + x_{a,j,i,t}). \tag{5.1}$$

Here, $x_{a,i,j,t}$ is the binary decision variable that describes if edge (i, j) is used by train a at time t. Parked trains do not consume any resources; therefore, self-edges are not included in the objective. The other binary decision variables in the model are $p_{a,i,t}$, which describes if train a is at node i at time t, and $y_{a,t}$, which describes

if train a is at the yard at time t.

$$p_{a,ag(a),at(a)} = 1 \quad \forall a \in A \tag{5.2}$$

$$\sum_{i \in TR} p_{a,i,t} = y_{a,t} \quad \forall a \in A, \forall t \in \{at(a), \ldots, T_{\max} - 1\} \tag{5.3}$$

$$\sum_{a \in A} p_{a,i,t} \leq 1 \quad \forall i \in TR, \forall t \in \{0, \ldots, T_{\max} - 1\} \tag{5.4}$$

$$\sum_{i,j \in c} \sum_{a \in A} (x_{a,i,j,t}) \leq 1 \quad \forall c \in C, \ \forall t \in \{0, \ldots, T_{\max} - 1\} \tag{5.5}$$

$$p_{a,i,t} - (y_{a,t} - y_{a,t+1}) \leq \sum_{j:(i,j) \in E} x_{a,i,j,t} \quad \forall a \in A, \ \forall i \in TR, \ \forall t \in \{at(a), \ldots, T_{\max} - 1\} \tag{5.6}$$

$$p_{a,j,t+1} = \sum_{i:(i,j) \in E} x_{a,i,j,t} \quad \forall a \in A, \ \forall j \in TR, \ \forall t \in \{at(a), \ldots, T_{\max} - 1\} \tag{5.7}$$

$$\sum_{(i,t) \in D(\theta(a), at(a) < t)} p_{a,i,t} = 1 \quad \forall a \in A \tag{5.8}$$

$$y_{a,t+1} \leq 1 - p_{a,i,t} \quad \forall a \in A, \forall (i,t) \in D(\theta(a)) \tag{5.9}$$

$$y_{a,t+1} \leq y_{a,t} \quad \forall a \in A, \forall t \in \{at(a), \ldots, T_{\max} - 2\} \tag{5.10}$$

To further define the model, we define constraints based on the TUSP inputs and requirements. A train a arrives at its arrival time $at(a)$ and track $ag(a)$, which we model through Constraint 5.2. While a train is at the yard, it has to be at a node, modeled through Constraint 5.3. To avoid collisions, we define Constraints 5.4 and 5.5, which ensure a node has at most one train parked on it at any time and that no two trains use conflicting edges at the same time, respectively. Conflicting edges are edges that cannot be used at the same time, as described in Section 4.4. The movement of the trains is modeled through Constraints 5.6 and 5.7. Constraint 5.6 concerns a train a leaving node i at time t through edge (i, j) and Constraint 5.7 concerns train a arriving at node j at time $t + 1$. In Constraint 5.6, the left-hand side becomes 0 if a train departs at time t due to the $-(y_{a,t} - y_{a,t+1})$ term. The inclusion of this term in the constraint avoids the need for a train to select an edge when it departs. The matching in the model is performed via Constraint 5.8, which ensures that each train is matched to exactly one departure with a matching type and a departure time later than the train's arrival time. Due to Constraint 5.4, only one train can be matched to each departure because otherwise two trains would be at the same node at the same time. Modeling the matching through this constraint implicitly solves the matching as part of the routing decisions. Constraint 5.9 ensures that when a train reaches a departure at time t, it is no longer present at

time $t+1$, and Constraint 5.10 ensures that a train cannot return to the yard after departure.

5.2 Lagrangian Relaxation

In this section, we explain how we apply standard LR to the MILP model. The main idea behind applying LR to the TUSP is that LR allows us to split the TUSP into multiple per-train sub-problems. To achieve this, we have to relax Constraints (5.4) and (5.5), which are the only coupling constraints that connect the different trains. These constraints are relaxed by incorporating them into the objective function. A term is added for both constraints and two Lagrangian multipliers are introduced, $\lambda_{i,t}$ for Constraint (5.4) and $\mu_{c,t}$ for Constraint (5.5), respectively. We thus have a different λ value for each node i at each time step t and a different μ value for each conflict set c at each time step. The Lagrangian objective function is then Equation (5.11).

$$L(\lambda,\mu) = \min \left(\begin{array}{l} \text{obj}(x) + \sum_{t=0}^{T_{\max}-1} \sum_{i \in TR} \lambda_{i,t} \left(\sum_{a \in A} p_{a,i,t} - 1 \right) \\ \quad + \sum_{t=0}^{T_{\max}-1} \sum_{c \in C} \mu_{c,t} \sum_{i,j \in c} (\sum_{a \in A} x_{a,i,j,t} - 1) \end{array} \right) \tag{5.11}$$

Next to minimizing the total movement, the objective is now also to minimize the penalties that are applied through $\lambda_{i,t}$ and $\mu_{i,j,t}$. The second term adds a penalty if more than one train is at the same node at the same time, and the third adds a penalty if two or more trains use edges part of the same conflicting edge set.

We can now decompose the problem into a series of independent subproblems, multipliers ($\lambda_{i,t} \cdot -1, \mu_{i,j,t} \cdot -1$) are constants and can thus be left out, resulting in the objective function for train a in Equation (5.12).

$$L_a(\lambda,\mu) = \min \left(\begin{array}{l} \sum_{t=0}^{T_{\max}-1} \sum_{\substack{(i,j) \in E \\ i<j}} (x_{a,i,j,t} + x_{a,j,i,t}) + \sum_{t=0}^{T_{\max}-1} \sum_{i \in TR} \lambda_{i,t} p_{a,i,t} \\ \quad + \sum_{t=0}^{T_{\max}-1} \sum_{c \in C} \mu_{c,t} \sum_{i,j \in c} (x_{a,i,j,t}) \end{array} \right) \tag{5.12}$$

To simplify future equations, we define the first term in Equation (5.12) as $\text{obj}_a(x)$, the second term as $\text{Ppen}_a(p)$, and the third term as $\text{Xpen}_a(x)$. The decomposition turns the problem into a shortest path problem for each train individually. In the LR approach, we solve the sub-problem (5.12), subject to Constraints 5.2,5.3,5.6-5.10, for each train individually and then update the Lagrangian multipliers through the subgradient method, which is explained in Section 2.2. The subgradient method

for the two penalty terms can be seen in Equations (5.13) and (5.14), where k is the iteration, starting at 1.

$$\lambda_{i,t}^{k+1} = \max(0, \lambda_{i,t}^{k} + \frac{1}{k}(\sum_{a \in A} p_{a,i,t} - 1)) \quad \forall i \in TR,\ \forall t \in \{0, \ldots, T_{\max} - 1\} \tag{5.13}$$

$$\mu_{c,t}^{k+1} = \max(0, \mu_{c,t}^{k} + \frac{1}{k}(\sum_{i,j \in c} \sum_{a \in A} x_{a,i,j,t} - 1) \quad \forall c \in C,\ \forall t \in \{0, \ldots, T_{\max} - 1\} \tag{5.14}$$

The Lagrangian penalties increase when multiple trains occupy the same node or use edges from the same conflicting edge set at the same time. They remain unchanged when exactly one train occupies a node or uses such an edge, and decrease when no train uses the node or edge. After updating the Lagrangian multipliers, we solve the subproblems again; this process is repeated until a feasible solution is found. If no feasible solution is found, the algorithm can be stopped when a time limit has been reached, when only a set number of conflicts remain, or when both conditions are satisfied. Stopping the algorithm when only a set number of conflicts remain can be useful when using the unfeasible solution found by LR as input for another algorithm. For example, as an initial solution for the Local Search (LS), this is not researched in this thesis.

5.3 Limitations of Lagrangian Relaxation: Detours and Symmetry

In this section, we answer sub-question **SQ3** for the standard LR approach by describing its main limitations. During initial testing on self-made instances, the LR approach was unable to solve two types of instances. First, we discuss instances in which the solutions require at least one train to diverge from its shortest path, We call this the detour problem. Second, we discuss instances that have many symmetric solutions due to symmetry in the shunting yards and trains.

5.3.1 Detour Problem

To illustrate the detour problem, we created a simple shunting yard, which we transformed into the graph structure, see Figure 5.1. A scenario for which the LR approach cannot find a feasible solution is described in Table 5.2. In this scenario, two trains have to switch positions, which is only possible if one train takes a detour. The SNG train departs one minute later, making the instance asymmetric; otherwise, it would belong to both problematic instance types. The cost of the shortest path for both trains is 2, whereas the cost of the alternative detour path is 3. There are multiple shortest paths for both trains due to the different time steps. They can move to node 3 at time 0 or time 1, and when moving to node 3 at time 0, they can wait one timestep at node 3. For the LR approach to find a feasible

solution, the detour path must become cheaper than all these shortest paths for one train.

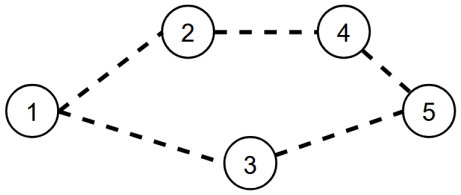


Figure 5.1: Simple shunting yard as graph.

Train	Arrival (i, t)	Departure (i, t)
SLT	(1, 0)	(5, 3)
SNG	(5, 0)	(1, 4)

Figure 5.2: Train arrival and departure track (i) and time (t) for the example scenario.

In the LR approach, we solve the shortest path problem for both trains separately. Suppose both trains select the shortest path in which they move to node 3 at time 1. A conflict then occurs at node 3 at time 1 since both trains arrive at node 3 at time 1. The penalty $\lambda_{3,1}$ is then set to 1 through the subgradient method (Equation (5.13)). Since the LR has not found a feasible solution, the shortest path problems are solved again.

For both trains, another shortest path, for example, moving to node 3 at time 1, is still cheaper than taking a detour. This leads to a conflict at node 3 at time 2. At this point, the penalty values are updated again. The penalty $\lambda_{3,1}$ is reduced to 0.5 and the penalty $\lambda_{3,2}$ is set to 0.5. The key point is that penalties do not accumulate on a single path. Since trains alternate between multiple shortest paths, a different conflict is penalized each iteration. As a result, all shortest paths for a train never become more expensive than the detour path since the maximum penalty applied to any conflict is 1.0 in the first iteration and decreases with each iteration.

The proposed solution to the detour problem is setting the cost of each movement to 0.01 instead of 1. This way, the penalties applied through LR, which typically start with 1.0, are actually high enough to push the trains to select different paths. By setting the cost to 0.01, we still aim to reduce the total movements but not at the cost of conflicts. Decreasing the cost is equivalent to increasing the step size of the subgradient method.

5.3.2 Symmetry

To illustrate why the LR approach struggles with symmetry, we created another simple shunting yard and transformed it into the graph structure, see Figure 5.3. In this yard, the gateway track is node 1. This yard is inspired by the Kleine Binckhorst shunting yard depicted in Figure 4.2. The gateway track in the Kleine Binckhorst is connected to nine tracks, which we reduced to three tracks in this example. Since the cost of each edge to these tracks is equal, an arriving train has three symmetric paths to choose from.

Consider a scenario in which three trains have to be parked at the shunting yard in Figure 5.3. Due to the symmetry, the trains may pick the same path, for example, the path to node 2. This leads to conflicts on the path to node 2, which will become more expensive in the next iteration through the penalty terms. In this iteration, each train will select a different path, either to node 3 or to node 4. Assume they all select node 3, which increases the penalty for node 3 in the next iteration. This leads to all trains picking the path to node 4 in the next iteration. For this example, the trains continue to select identical paths since the paths and trains are symmetric. This prevents the LR approach from converging to a feasible solution, even for a low number of trains.

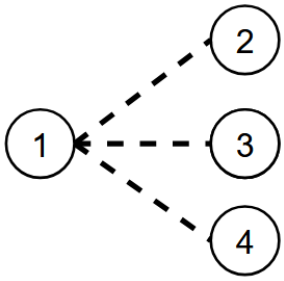


Figure 5.3: Simple shunting yard as graph.

For the simple shunting yard, adding some randomness in the cost of each path for each train can break this symmetry. However, for the Kleine Binckhorst, where we have nine symmetric paths and trains, this randomness is not enough to break the symmetry consistently, which means the LR never finds feasible solutions for larger symmetric instances.

Another form of symmetry comes from the matching; if two or more trains have the same type, they can pick the same departing train to match, which will cause similar behavior as for the paths described above. Even for the example given for the detour problem above, these symmetry problems occur when setting the departure time equal for the two trains. The proposed solution to these symmetry problems is to use Augmented Lagrangian Relaxation and the Alternating Direction Method of Multipliers, which will be described in the next section.

5.4 Augmented Lagrangian Relaxation & the Alternating Direction Method of Multipliers

To solve the symmetry problems that occur for the LR approach, we extend it with Augmented Lagrangian Relaxation (ALR), which introduces a quadratic penalty term for each relaxed constraint in the objective. These quadratic penalty terms couple the trains, which solves the symmetry because a path selected by one train

becomes more expensive for another train. However, these quadratic terms destroy the decomposability of the LR. The Alternating Direction Method of Multipliers (ADMM) is used to regain the separability in the resulting problem.

For the ALR, we introduce a quadratic term for the relaxed Constraints 5.4 and 5.5 in addition to the Lagrangian penalty terms. The objective function for ALR is then Equation (5.15), with penalty parameter ρ. The last two terms in Equation (5.15) have been added compared to the LR objective function in Equation (5.11).

$$L(\lambda,\mu,\rho) = \min \left(\begin{array}{l} \text{obj}(x) + \sum_{t=0}^{T_{\max}-1} \sum_{i \in TR} \lambda_{i,t} \left(\sum_{a \in A} p_{a,i,t} - 1 \right) \\ \quad + \sum_{t=0}^{T_{\max}-1} \sum_{c \in C} \mu_{c,t} \sum_{i,j \in c} (\sum_{a \in A} x_{a,i,j,t} - 1) \\ \quad + \sum_{t=0}^{T_{\max}-1} \sum_{i \in TR} \frac{\rho}{2} \max \left(0, \sum_{a \in A} p_{a,i,t} - 1 \right)^2 \\ \quad + \sum_{t=0}^{T_{\max}-1} \sum_{c \in C} \frac{\rho}{2} \max \left(0, \sum_{i,j \in c} \sum_{a \in A} x_{a,i,j,t} - 1 \right)^2 \end{array} \right) \tag{5.15}$$

To split the objective function into a sequence of interdependent subproblems, we use ADMM. This is done by regrouping the ALR terms in Equation (5.15) into terms of the current train a and the terms of other trains, and by rewriting the LR terms as before. This results in the objective function in Equation (5.16) for train a.

$$L_a(\lambda,\mu,\rho) = \min \left(\begin{array}{l} \text{obj}_a(x) + \text{Ppen}_a(p) + \text{Xpen}_a(x) \\ + \sum_{t=0}^{T_{\max}-1} \sum_{i \in TR} \frac{\rho}{2} \max \left(0, p_{a,i,t} + \sum_{a' \in A \setminus \{a\}} p_{a',i,t} - 1 \right)^2 \\ + \sum_{t=0}^{T_{\max}-1} \sum_{c \in C} \frac{\rho}{2} \max \left(0, \sum_{i,j \in c} \left(x_{a,i,j,t} + \sum_{a' \in A \setminus \{a\}} x_{a',i,j,t} \right) - 1 \right)^2 \end{array} \right) \tag{5.16}$$

To simplify this equation, let ${\eta^a}_{i,t}$ be the number of trains in $A \setminus \{a\}$ that use node i at time t and $\varphi^a_{c,t}$ be the number of trains in $A \setminus \{a\}$ that use an edge in conflict set c at time t:

$$\eta^a_{i,t} = \sum_{a' \in A \setminus \{a\}} p_{a',i,t} \quad \forall i \in TR \tag{5.17}$$

$$\varphi^a_{G,t} = \sum_{a' \in A \setminus \{a\}} (x_{a',i,j,t}) \quad \forall i,j \in c. \tag{5.18}$$

Substituting these into the objective function in Equation (5.16) results in the objective function in Equation (5.19) for train a.

$$L_a(\lambda, \mu, \rho) = \min \left(\begin{array}{l} \text{obj}_a(x) + \text{Ppen}_a(p) + \text{Xpen}_a(x) \\ + \sum_{t=0}^{T_{\max}-1} \sum_{i \in TR} \frac{\rho}{2} \max\left(0, p_{a,i,t} + \eta^a_{i,t} - 1\right)^2 \\ + \sum_{t=0}^{T_{\max}-1} \sum_{c \in C} \frac{\rho}{2} \max\left(0, \sum_{i,j \in c} \left(x_{a,i,j,t} + \varphi^a_{i,j,t} - 1\right)\right)^2 \end{array} \right) \tag{5.19}$$

In the ADMM approach, we solve the MILP problem with Objective (5.19) subject to Constraint 5.2,5.3,5.6-5.10 for each train sequentially. The $\eta^a_{i,t}$ and $\varphi^a_{G,t}$ values for train a are computed before a sub-problem is solved. The Lagrangian multipliers are updated after all sub-problems have been solved. This process is repeated until a feasible solution is found. As with LR, if no feasible solution is found, the algorithm can be stopped when a time limit has been reached, when only a set number of conflicts remain, or when both conditions are satisfied. In the next section, we describe how these sub-problems can be formulated as shortest path problems.

5.5 Shortest Path Formulation

The sub-problems can be solved using shortest path algorithms [Dijkstra, 1959]. The time-extended graph formulation described in Section 4.4 already contains most elements required to solve the sub-problems with shortest path algorithms. However, in the current formulation, each train has a single goal node, while when considering the matching, a train can match to multiple departures, resulting in multiple feasible goals. This formulation thus has to be extended such that the matching is properly modeled. Furthermore, the LR and ALR penalty terms need to be incorporated into the path costs.

The gateway track cannot be used as a unique goal node, since trains are only allowed to depart at specific times and can match to multiple departures. Therefore, we add a sink node to the graph, which becomes the goal node. We then create a different graph for each train a, in which for each departure with matching train type and a departure time after the train's arrival time ($(i,t) \in D(\theta(a), at(a) < t)$), we add the edge $(i, sink, t)$. These edges have no traversal time. For example, if there are two departures with type x at time $t = 4$ and at $t = 9$, then each train with type x has an edge from the gateway track to the sink node at $t = 4$ and at $t = 9$.

To include the LR and ALR penalties in the shortest path formulation, we add them to the edge costs. The edge cost for an edge (i, j, t) is given by Equation 5.20.

$$\begin{aligned}\text{edge\_cost}_{(i,j,t)} = & \ 0.01 + \lambda_{j,t+1} p_{a,i,t} + \frac{\rho}{2} \max(0, p_{a,i,t} + \eta^a_{j,t+1} - 1)^2 \\ & + \sum_{c \in C \mid (i,j) \in c} (\mu_{c,t} x_{a,i,j,t} + \frac{\rho}{2} \max(0, x_{a,i,j,t} + \varphi^a_{i,j,t} - 1)^2)\end{aligned} \tag{5.20}$$

This equation can be simplified by considering the cases where the decision variables are 0 or 1. When they are 0, the edge is not used by train a, and there is thus no additional cost for the path. When they are 1, substituting them into Equation 5.20 yields Equation 5.21. This equation can be simplified to Equation 5.22, as the 1 and -1 cancel out and the max can be left out since $\eta^a_{j,t+1}$ and $\varphi^a_{i,j,t}$ are always larger than or equal to 0. The cost of an edge thus includes the base cost (0.01), the LR and ALR penalties for being at node $(j, t+1)$, and the LR and ALR penalties for using edge (i, j, t).

$$\begin{aligned}\text{edge\_cost}_{(i,j,t)} = & \ 0.01 + \lambda_{j,t+1} + \frac{\rho}{2} \max(0, 1 + \eta^a_{j,t+1} - 1)^2 \\ & + \sum_{c \in C \mid (i,j) \in c} (\mu_{c,t} + \frac{\rho}{2} \max(0, 1 + \varphi^a_{i,j,t} - 1)^2)\end{aligned} \tag{5.21}$$

$$\text{edge\_cost}_{(i,j,t)} = 0.01 + \lambda_{j,t+1} + \frac{\rho}{2} (\eta^a_{j,t+1})^2 + \sum_{c \in C \mid (i,j) \in c} (\mu_{c,t} + \frac{\rho}{2} (\varphi^a_{i,j,t})^2) \tag{5.22}$$

The cost of self-edges is defined by Equation 5.23, which only includes the LR and ALR node penalties.

$$\text{edge\_cost}_{i,i,t} = \lambda_{i,t+1} + \frac{\rho}{2} (\eta^a_{i,t+1})^2 \tag{5.23}$$

5.6 Psuedocode ADMM

The pseudocode for the ADMM approach can be found in Algorithm 1. In lines 1 through 4, the yard and scenario data are stored, and a graph is created for each train based on which departures it can match to, as explained in Section 5.5. The main loop of the algorithm starts at line 6 and ends when either a feasible solution is found or when the time out is reached. In lines 7 through 10, for each train, the shortest path problem is solved with the A* algorithm after computing the ALR penalties based on the solutions of the other trains and setting the edge cost in the graph. The ALR penalties are computed following Equations 5.17 and 5.18 and the edge costs are set following Equations 5.20 and 5.23. After a path is found for each train, the combined paths are checked for conflicts in line 11. Finally, in lines 12 through 16, the Lagrangian penalties are set following Equations 5.13 and 5.14, based on the conflicts and iteration k.

Algorithm 1 Pseudocode ADMM

```
1:  nodes, edges, conflict_groups = load_location("yard")
2:  arrival_trains, departures, time_window = load_scenario("scenario")
3:  for a in arrival_trains do
4:      graphs[a] = create_graph_agent(a, departures, nodes, edges)
5:  k = 1
6:  while not solution_found and not time_out do
7:      for a in arrival_trains do
8:          ALR_penalties = update_ALR_penalties(paths)
9:          graphs[a] = set_cost(graphs[a], LR_penalties, ALR_penalties)
10:         paths[a] = astar_shortest_path(graphs[a])
11:     conflicts, solution_found = compute_conflicts(paths)
12:     for t in time_window do
13:         for i in nodes do
14:             update_LR_penalties_node(k, t, i, conflicts(t, i))
15:         for g in conflict_groups do
16:             update_LR_penalties_conflict_group(k, t, g, conflicts(t, g))
17:     k += 1
```

Chapter 6

Experimental Evaluation

The main goals of this thesis are to determine whether Lagrangian Relaxation (LR) methods can be applied to the Train Unit Shunting Problem (TUSP) and how their performance compares to that of the state-of-the-art Local Search (LS) approach. In this chapter, we aim to answer these questions with an experimental evaluation of the methods described in Chapter 5.

In Section 6.1, we describe the experimental setup in which we list the computational details, and discuss the characteristics of the shunting yard and of the scenarios used to evaluate the methods. We then discuss the performance of the Mixed Integer Linear Programming (MILP) approach in Section 6.2 to show that the model cannot be solved with exact methods within a realistic time for use in practical planning operations. This is followed by a discussion of the performance of the standard Lagrangian Relaxation (LR) approach in Section 6.3. In Section 6.4, we discuss hyperparameter tuning for the Alternating Direction Method of Multipliers (ADMM) approach. We then discuss experiments that compare LS to the ADMM approach for different number of train types and for different time windows in Section 6.5. The performance is measured in terms of the number of feasible solutions found and the computation time. In these experiments, we test two versions of LS, one with continuous tracks and the other with discrete tracks. These versions are described in Section 6.1.3.

6.1 Experimental Setup

In this section, we describe the setup for the experiments. We begin with an overview of the computational details for the different methods and the experiments. Next, we describe how the shunting yard on which we test the different approaches is turned into the graph structure described in Section 4.4. We then discuss the difference between the continuous and discrete LS. Finally, we describe the characteristics of the TUSP scenarios on which we evaluate the different approaches.

6.1.1 Computational details

All experiments in this chapter were run on the DelftBlue supercomputer [Delft High Performance Computing Centre , DHPC]. Each experiment was run with an Intel(R) Xeon(R) Gold 6226R CPU on one core with 3968 MB. The MILP and LR approaches were programmed in Python 3.10 [Python Software Foundation, 2021] using Pyomo [Hart et al., 2017] and solved with Gurobi [Gurobi Optimization, LLC, 2024]. ADMM was also programmed in Python 3.10 and solved with the A* algorithm using rustworkx [IBM Quantum Developers, 2023]. In this implementation, no parallelization was used. To solve the Detour problem described in Section 5.3.1, the cost of a movement is set to 0.01 for LR and ADMM. The implementations of the different approaches can be found on the Robust-Rail mathematical models GitHub [Robust-Rail-NL, 2026a].

The LS algorithm was programmed in C# and can be found on GitHub [Robust-Rail-NL, 2026c]. This implementation is based on van den Broek [2016]. It has two stages: Tabu Search and Simulated Annealing. Tabu Search was run for 200 iterations, after which Simulated Annealing was run until a feasible solution was found or the timeout was reached. Each algorithm uses 1 as the random seed.

6.1.2 Shunting Yard

The shunting yard on which we test our implementation is the Kleine Binckhorst in Den Haag, depicted in Figure 4.2. The Kleine Binckhorst was selected as the shunting yard, as TUSP scenarios there appear to be on the border of the computational limits of the LS algorithm. Experimenting on different yards would provide more insights, but this is outside of the scope of this thesis. For the MILP, LR, and ADMM approaches, the Kleine Binckhorst is turned into the graph structure described in Section 4.4. The resulting graph can be seen in Figure 6.1. The Kleine Binckhorst has one gateway track, which is represented by node 0. The gateway track is connected to nine tracks through a series of switches. These switches are represented by nine edges from node 0 to nodes $1-9$. All tracks on which parking is allowed are split into multiple nodes based on their length, and an edge is added between them. To keep the figure clear, $T1$ and $T2$ were added. Each edge entering one of these nodes represents multiple edges that connect to the endpoints of the edges leaving the node on the other side. In the actual graph, these nodes are absent, and the incoming and outgoing edges represent direct connections between their endpoints. Parking is not allowed on red nodes; reversals are allowed on all nodes.

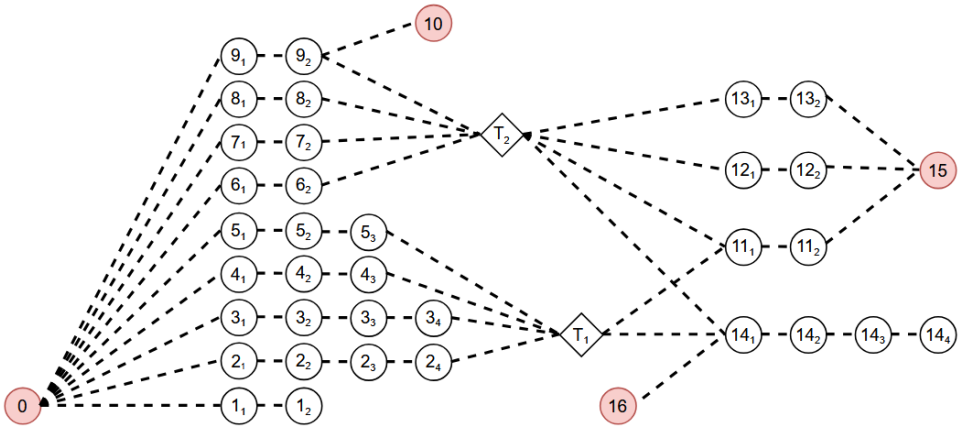


Figure 6.1: Graph representation of the Kleine Binckhorst shunting yard in Den Haag.

In the model described in Section 5.1, multiple trains are allowed to move at the same time as long as they do not use edges that are part of the same conflicting edge set. In practice, it can be that only one train is allowed to move at the same time or that only one driver is available. Furthermore, it is also not possible in the LS algorithm to allow trains to move at the same time. To ensure a fair comparison between ADMM and LS, and to ensure that the problem is realistic, we thus only allow one movement at the same time. We model this for the MILP, LR, and ADMM approaches by having one conflicting edge set at each time step, namely the set of all non-self-edges. Constraint 5.5 is then practically the same as the following constraint:

$$\sum_{\substack{(i,j)\in E \\ i\neq j}} \sum_{a\in A} (x_{a,i,j,t}) \leq 1 \quad \forall t \in \{0,\ldots,T_{\max}-1\}. \tag{6.1}$$

For an overview of the conflicting edge sets in case multiple movements at the same time would be allowed, see Appendix 8.

6.1.3 Continuous & Discrete Local Search

In this chapter, ADMM is compared to two versions of the LS: continuous and discrete. In the continuous version, tracks have continuous length, and a move from the A-side to the B-side of a track takes no time. A track can thus fit multiple trains. In the discrete version, tracks are split into multiple tracks of 100 meters with switches in between them. These switches only connect two tracks, and traversing them takes one minute. The continuous version thus uses Figure 4.2 as yard layout, whereas the discrete version uses Figure 6.1.

The reason for comparing ADMM to these two versions is the fact that in the

MILP model, the decision was made to split tracks into multiple nodes. This decision adds extra traversal time to the TUSP that is not present in practice. In the time aspect of the TUPS, ADMM thus solves a different version than in practice. By comparing the continuous LS version to both ADMM and the discrete version, we can analyze the effect that splitting tracks into multiple nodes has on the solvability of the TUSP. The discrete LS version allows us to compare ADMM and LS on the same version of the TUSP problem. This comparison will show the difference in solving capability of ADMM and LS, independent of modeling choices and problem definition.

6.1.4 Scenarios

The scenarios on which the different algorithms are tested were generated using the Robust-Rail Generator [Robust-Rail-NL, 2026b]. The scenarios generated by the Generator are not guaranteed to be feasible. Since incorporating the length of trains is outside of the scope of this thesis, each train has a length of 100 meters, which is the average length of NS train units [OV in Nederland Wiki, NVBS]. The number of train types in a scenario can have a significant impact on the number of feasible solutions of a scenario. When all trains have the same type, each train can be matched to each departure, whereas if all trains have a different type, their departure is fixed. In the scenarios, it is not the case that the number of trains of different types is equal in a given scenario. There may be only one train of a type, whereas there could be multiple trains of another type.

The goal of the experiments is to compare how the different algorithms perform for scenarios in which a large part of the yard is occupied. In practice, about 80% of a yard is occupied, which means $27/28$ trains at the Kleine Binckhorst when all trains are 100 meters long. There is space for 34 trains on the Kleine Binckhorst. In the full TUSP problem, certain tracks at a yard are used for servicing. The inclusion of service tasks means fewer tracks are always available for parking. Since servicing is not considered in this thesis, we test scenarios up to 33 trains, as there is more space available for parking without servicing. To ensure that 33 trains are actually present at the yard at the same time, all trains arrive before a train departs.

The time window of the scenarios has a significant impact on the feasibility of scenarios and the number of feasible solutions of a scenario. A larger time window allows for more time to reallocate trains and increases the solution space. In practice, a scenario can take around 4 hours for shunting during the day, 7 or more hours during the night, or in some cases 24 hours. In these scenarios, trains also have to undergo service tasks, be (de)coupled, and reverse their direction. These components of the TUSP are not considered in this thesis, but they use a significant part of the time window. Furthermore, the decision to split tracks into multiple nodes also affects the time needed to perform train movements at the yard. It is therefore difficult to determine a realistic time window from practice for the TUSP

problem solved in this thesis. In Section 6.5.2, we therefore illustrate the effect the time window has on the performance of LS and ADMM. In the other experiments, a fixed time window will be used; the exact time window will be specified and motivated for each experiment.

6.2 Mixed Integer Linear Programming Approach

Can the MILP model solve the simplified version of the TUSP in a reasonable time?

The expectation is that the MILP model cannot be solved with exact methods in a reasonable time, as supported by previous research. We therefore analyze its performance on instances with up to 20 trains at most, 5 types, and a time window of 80 minutes. In these scenarios, the relatively low number of trains and types, and a small time window, all contribute to a relatively lower complexity of the problem. The instances in this experiment are thus relatively simple. We use a timeout of 20 minutes and test a total of 80 instances, 20 for each number of trains.

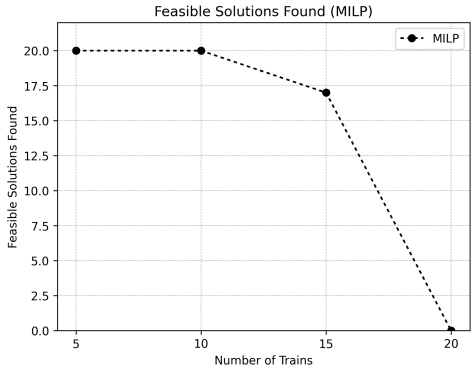


Figure 6.2: MILP

The results of this experiment can be seen in Figure 6.2. As can be seen in the figure, MILP cannot solve scenarios with 20 trains. We also tested the MILP model with more cores, and even with 48 cores, it still required 20 minutes to solve scenarios with 20 trains. This result shows that even on a simplified version of the TUSP, the exact method is unable to solve the TUSP within a realistic time, and that the simplified version is still difficult to solve. This was an important motivation to study LR, of which the evaluation is discussed in the next section.

6.3 Lagrangian Relaxation Approach

Can standard LR find feasible solutions for the simplified version of the TUSP?

As discussed in Section 5.3.2, LR struggles with solving symmetric instances. The graph representation of the Kleine Binckhorst has significant symmetry in it. Nodes $1-9$, for example, are all symmetric, and each train thus has at least nine symmetric shortest paths. The LR algorithm was unable to solve scenarios with 10 trains, so we did not run any experiments. To answer sub-questions **SQ2** and **SQ3**: standard LR cannot find feasible solutions for TUSP instances of significant size due to symmetry.

In the next section, the hyperparameter tuning for ADMM is discussed, which is the method used in this thesis to solve these symmetry issues. The main questions regarding ADMM are: What are the best hyperparameters for ADMM, does ADMM solve the symmetry issues that occur with standard LR, and how does its performance compare to the LS approaches?

6.4 ADMM Hyperparameter Tuning

In this section, we discuss the experiments that were used to determine the best ρ and the subgradient starting stepsize value for ADMM. In the subgradient method, the N in N/k can be changed, where $N = 1$ is the standard value. Here N is the starting stepsize. In these experiments, we compare the number of feasible solutions found by ADMM across different values of ρ and N. We start with experiments to find the optimal ρ value with the starting stepsize fixed at the standard value of 1, followed by experiments to identify the best starting stepsize value corresponding to this ρ value. In these experiments, we take the average over 100 scenarios and use a timeout of 30 minutes. These scenarios all have 30 trains, 5 different train types, and a 7 hour time window. These parameters were chosen as preliminary experiments showed that ADMM was unable to find solutions for about 30% of scenarios at these values. Although the performance of the ρ values might vary at different values for these parameters, we expect that these effects are minimal. Researching this further is also outside of the scope of this thesis and of minimal relevance.

The hypothesis is that the performance of ADMM depends on the ρ value, the starting stepsize value, and the movement cost. These values determine the interaction between the cost of conflicts, the influence of the Lagrangian penalties, and whether taking a detour is preferred over conflicts. As the main goal is to find feasible solutions, we set the movement cost to 0.01 such that taking a detour is preferred over conflicts. To determine the best ρ and starting stepsize value, we experiment with different values.

The expectation is that at a lower ρ value, the search process will be dominated by the Lagrangian penalties, leading to a lower number of feasible solutions as the symmetry issues will be more prevalent. Increasing the ρ value is expected to

increase the impact of the augmented terms, while decreasing the impact of the Lagrangian penalties. The best performance is expected at values where these two factors are properly balanced.

What is the best performing rho value for ADMM?

We tested the following ρ values: $0.1, 0.5, 1, 1.5, 2, 3$. These values were selected based on preliminary experiments. The number of feasible solutions found by ADMM for these values can be seen in Figure 6.3. In this figure, we see that $\rho = 0.5$ was the best performing value. As expected, a low value (0.1) leads to fewer feasible solutions, and the number of feasible solutions found decreases after the optimal value $\rho = 0.5$. As $\rho = 0.5$ finds only one solution more than $\rho = 1.0$, we did not further research the number of feasible solutions found by ρ values between 0.1 and 1.

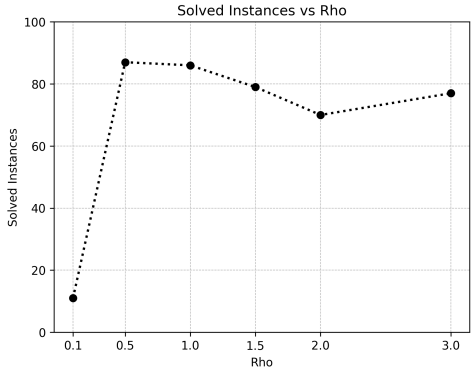


Figure 6.3: Number of feasible solutions found for different ρ values.

What is the best performing starting stepsize value for ADMM?

To determine what starting stepsize value provides the best performance, we experimented with the following values for N: $0.1, 0.5, 1, 1.5, 2, 3$. The number of feasible solutions found by ADMM for these values can be seen in Figure 6.4. In the figure, we see that the difference in the number of feasible solutions found is not significant. We therefore decided to do all future experiments with the standard value of 1. $\rho = 0.5$ and $N = 1$ will thus be used in all future experiments.

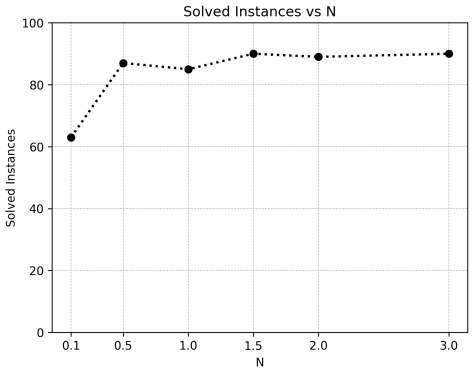


Figure 6.4: Number of feasible solutions found for different starting stepsize values.

Although we expect that the ρ and starting stepsize value have the most impact on the performance, we suspect that other characteristics of the subgradient method can also influence the performance. Namely, the decrease rate of the Lagrangian penalties over the iterations and the speed with which the penalties are decreased when a node or edge is not used.

The decrease rate of the Lagrangian penalties currently follows the series $1, \frac{1}{2}, \frac{1}{3}, \ldots$. It impacts the influence of the Lagrangian penalties in later iterations. As the size of the Lagrangian penalties decreases each iteration, so does their impact compared to the quadratic ADMM penalties, which are fixed.

The speed with which the penalties are decreased when a node or edge is not used is currently $\frac{-1}{k}$. This value can influence the impact of the Lagrangian penalties on the solving process. When a node or edge is not used at a point in time, its corresponding Lagrangian penalty is decreased. When a train has many equal cost paths that are all conflicting, we observe that the train cycles between these paths instead of finding another solution. The penalties on these paths might decrease too quickly, causing any of these paths to always be cheaper than a path that might lead to a feasible solution. This is similar behavior to that in the symmetry problem described in Section 5.3.2. Investigating these points is outside of the scope of this thesis, but could be interesting for future work.

6.5 Alternating Direction Method of Multipliers vs Local Search

In this section, we compare the performance of ADMM, continuous LS, and discrete LS. We answer sub-questions **SQ2–5**. Performance is evaluated on the number of feasible solutions found and the average computation time. We do two experiments. In the first experiment, we compare their performance for a fixed

time window of 7 hours and different numbers of train types. In the second experiment, we compare their performance for different time windows for 20 trains. The goal of the first experiment is to compare the performance for different numbers of trains and to see the effect of the number of types at a fixed time window. The goal of the second experiment is to research the impact of the time window on the performance, as instances with a smaller time window in general have fewer feasible solutions.

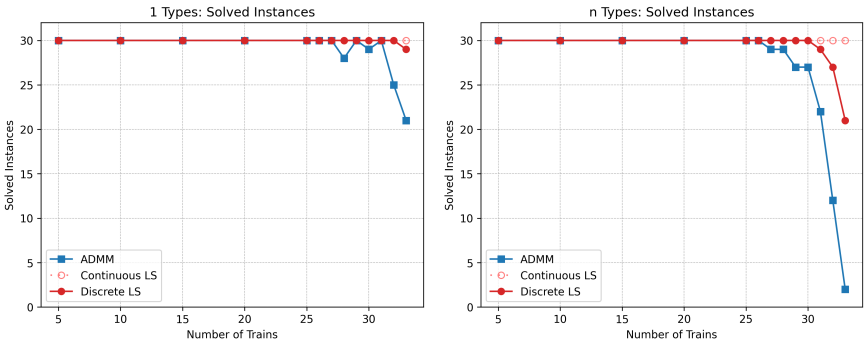
6.5.1 Fixed Time Window & Train Types

In this experiment, we compare the performance of the different approaches for different numbers of train types. Experimenting with different numbers of train types will illustrate the effect that the number of possible matchings has on the performance of the different approaches. We experiment with these numbers of train types: 1, 5, $n/3$, and n, where n is the number of trains. For 1 train type, each train can be matched to each departure, and for n train types, every train's departure is predetermined. 5 train types is the typical value that occurs at NS, and $n/3$ was added to further illustrate the effect of the number of types on the performance. The number of possible matchings and the number of feasible solutions thus decrease as the number of types increases.

The scenarios have a time window of 7 hours such that they have many feasible solutions. The number of feasible solutions decreases as the number of trains increases, as on average, each train can move less. We test 30 scenarios per combination of type and train numbers. In total, 1500 scenarios were run for each algorithm, with a timeout of 30 minutes.

Does ADMM outperform discrete/continuous LS in terms of feasible solutions found?

In Figures 6.5a and 6.5b, the number of feasible solutions found for ADMM, discrete LS, and continuous LS for 1 and n train types can be seen. We only show these results as they are the outer train type values; the results for the other number of train types can be found in Appendix 9. The trend in the number of train types is that, for more train types, fewer feasible solutions are found by both ADMM and discrete LS. In the figures, we see that continuous LS finds solutions for all scenarios, whereas discrete LS finds solutions for all scenarios up to 30 trains. For ADMM, we observe that it finds significantly fewer solutions than either of the LS versions. Out of the 1500 scenarios, discrete LS and ADMM were unable to find a solution for 27 and 175 scenarios, respectively.



(a) (b)

Figure 6.5: Number of feasible solutions found for 1 and n types, for ADMM, discrete LS, and continuous LS.

From these results, we conclude that the modeling choice to split tracks into multiple nodes has a significant impact on the complexity of the TUSP since the continuous LS outperforms both the discrete LS and the ADMM algorithm in terms of feasible solutions found. The reason for this is that moving a train from the A-side to the B-side of a track takes no time for the continuous LS, whereas it takes one minute per 100 meter for the discrete LS and ADMM approaches. Continuous LS, thus, has significantly more time to reallocate trains and therefore solves a simpler but more realistic version of the TUSP. The argument that TUSP scenarios at the Kleine Binckhorst appear to be on the border of what the LS approach can solve does not hold for the continuous LS, as the time window in these scenarios is too large to be realistic. In the further discussions and experiments, the continuous LS will not be included as its results can all be explained with the above reasoning.

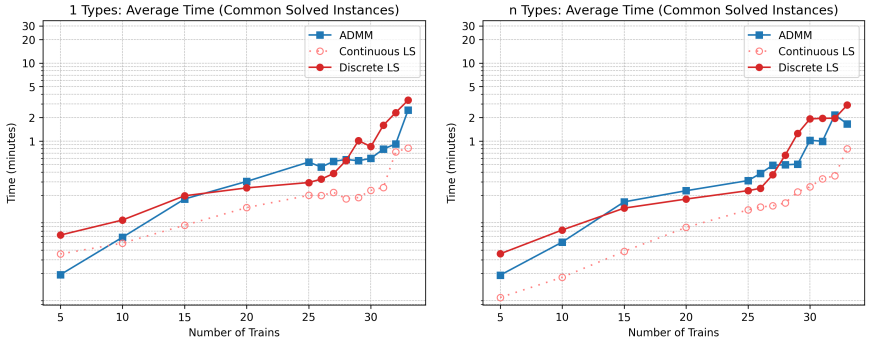
Furthermore, we conclude that ADMM solves the symmetry issues that occur for standard LR and can solve scenarios up to 25 trains consistently, but performs significantly worse than the discrete LS version for scenarios with more trains. We observe that, when ADMM does not find a feasible solution, it gets either stuck in a cycle or its solution no longer changes. As discussed in Section 6.4, we believe this happens because the Lagrangian penalties are not high enough in later iterations and do not accumulate enough to explore different solutions. In literature, these issues are often resolved by using a different penalty update method, such as the Polyak step size method [Polyak, 1969] or Surrogate LR [Bragin et al., 2015]. Another option for addressing these cycles could be to add Tabu Search to ADMM.

Solving problems with LR often provides lower bounds, but the solutions obtained are not always feasible. Often, primal recovery techniques [Sherali and Choi, 1996] are used to construct feasible solutions from the relaxed solutions. There are, on average, only 2 conflicts left when ADMM does not find a feasible solution, using primal recovery could thus be a logical choice in future work.

In the results, we also observe that having more train types leads to fewer feasible solutions found for both discrete LS and ADMM. This is expected since having more train types reduces the number of feasible matchings and thus the number of feasible solutions.

Does ADMM outperform discrete LS in terms of computation time?

In Figures 6.6a and 6.6b, the average computation time for ADMM and discrete LS for 1 and n types can be found. To remove bias, these figures include only scenarios for which all approaches found a feasible solution. These figures use a logarithmic scale. The results for the other number of train types are similar and can be found in Appendix 9. In these figures, we see that the average computation time of ADMM and discrete LS is similar. From these results, we can conclude that in terms of computation time, ADMM performs on par with the discrete LS.



(a) (b)

Figure 6.6: Average computation time of commonly solved instances on a logarithmic scale for 1 and n train types for ADMM, discrete LS, and continuous LS.

6.5.2 Time Window

In this experiment, we investigate the influence of the time window on the performance. We test the different approaches on scenarios with 20 trains, 5 train types, and a variable time window. Testing for different numbers of trains is outside the scope of this thesis. The scenarios have 20 trains, since preliminary results indicated that discrete LS struggles to find feasible solutions for 20 trains when the time window is small. Furthermore, 20 trains occupy only 60% of the yard, meaning these instances were initially expected to be relatively easy. The effect of the number of train types was already illustrated in the previous experiment, and we thus use 5 train types, which is the typical value that occurs at NS. For each time window, 30 scenarios were run, with a timeout of 30 minutes. We increase the time window starting from 4800 seconds, which is the lowest time window for

which ADMM was able to find solutions, up to 6000 seconds. These scenarios have significantly fewer feasible solutions than those in the previous experiment, where a time window of 7 hours (25200 seconds) was used. This experiment was not done for the continuous LS, as the lowest time window is still rather large for the continuous LS, and it would thus be able to solve all instances.

Does the time window impact the number of feasible solutions found by ADMM and discrete LS?

In Figure 6.7, the number of feasible solutions found for different time windows, for ADMM and discrete LS can be seen. These scenarios had 20 trains. In this figure, we see that ADMM finds significantly more feasible solutions than discrete LS as the time window decreases. In total, discrete LS was unable to find a feasible solution to 47 scenarios, whereas for ADMM, this was just the case for 4 instances.

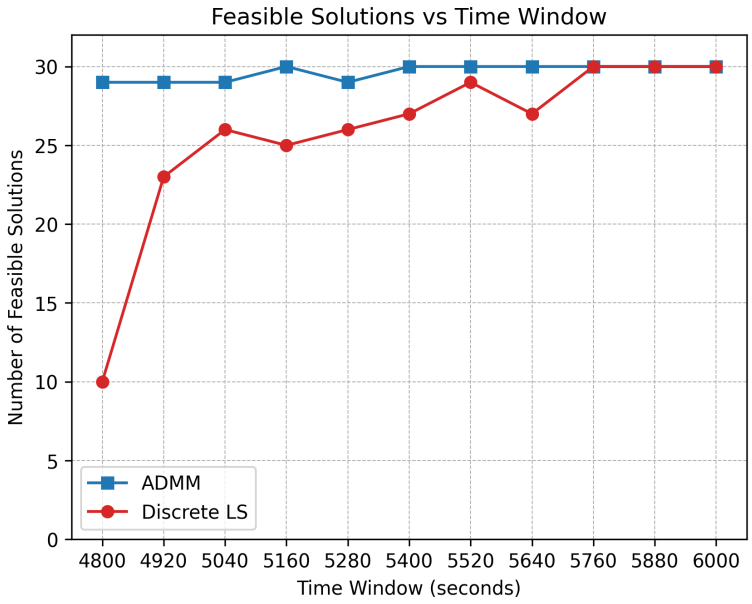


Figure 6.7: Number of feasible solutions found for ADMM and discrete LS, for different time windows.

From this result, we conclude that ADMM finds more feasible solutions for scenarios that are more time restricted than the discrete LS. We hypothesize that the LS algorithm struggles with scenarios that have a smaller time window, because they, in general, have fewer feasible solutions. As the problem space contains fewer feasible solutions, the LS is less likely to find one and is affected more by its initial solution. ADMM, on the other hand, is not affected by the smaller time window. We hypothesize that, when there is a large solution space, each train has more paths of equal cost. In that case, the LR penalties are not able to compound on certain

paths, making the algorithm get stuck in cycles more often.

Does the time window impact the computation time of ADMM and discrete LS?

In Figure 6.8, the average computation time for different time windows, for ADMM and discrete LS can be seen. To remove bias, this figure includes only scenarios for which both ADMM and discrete LS found a feasible solution. In this figure, we see that ADMM is significantly faster than discrete LS for all time windows. The spikes observed for ADMM in the figure are due to two instances where a feasible solution was found, but ADMM required substantially more time to converge.

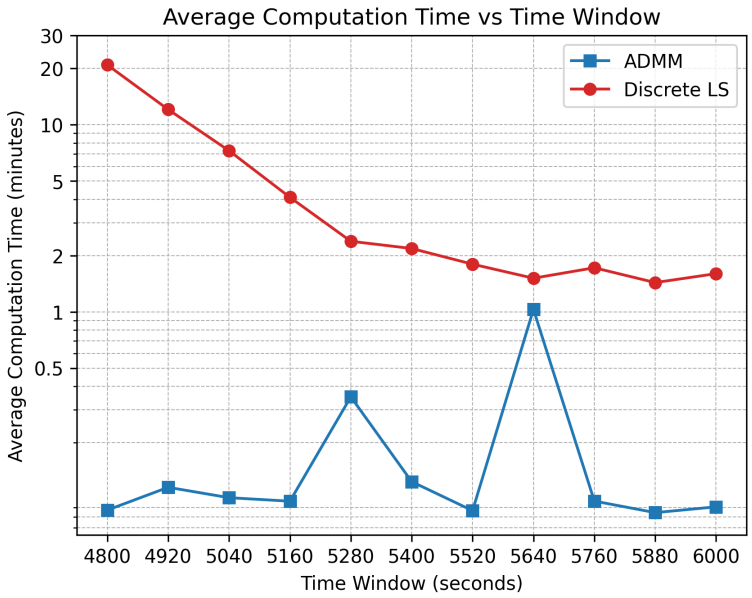


Figure 6.8: Average computation time of commonly solved instances on a logarithmic scale for ADMM and discrete LS, for different time windows.

These results show that for more time restricted instances, the ADMM approach is significantly faster than the discrete LS. The discrete LS is slower than for larger time windows. As there are fewer feasible solutions, it requires more time to explore the solution space before finding a solution.

6.6 Continuous ADMM Extension

In Section 6.5.1, we have seen that the modeling choice to split tracks into multiple nodes has a significant impact on the complexity of the TUSP. We therefore extend the ADMM approach such that it is continuous, which entails that traversal of the edges between nodes originating from the same track is instant. This version of the

TUSP is more realistic. In this section, we first explain how we extend the ADMM approach and then discuss the experimental evaluation of this extension. The time window experiment in Section 6.5.2 is not performed for continuous ADMM, as this was outside of the scope of this thesis.

6.6.1 Shortest Path Formulation Extension

To allow continuous movement on the same track, we add additional edges to the time-extended graph. For each track that is split into multiple nodes, we add two types of edges, internal and external. For each pair of nodes originating from the same track, we add an internal edge between them if no edges between these nodes exist yet. Traversing internal edges represents moving to a different part of the same track. For each edge connecting two nodes originating from different tracks, we add external edges between each pair of nodes that originate from different tracks, if no edges between these nodes exist yet. In Figure 6.9, an example of what edges are added to the graph for continuous ADMM on a part of the Kleine Binckhorst graph can be seen.

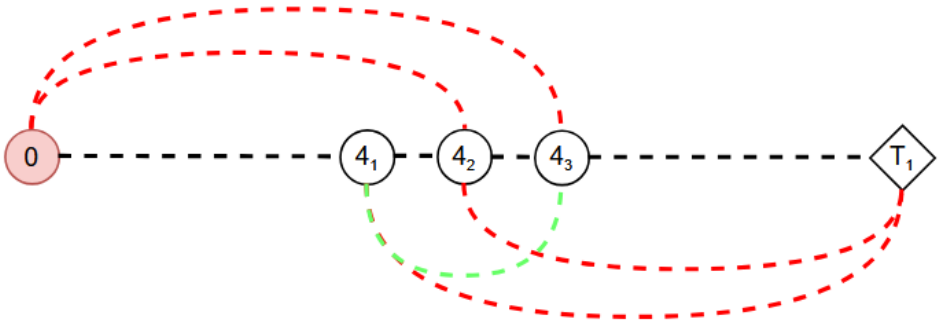


Figure 6.9: Internal (green) and external (red) edges that are added to the nodes originating from track 54 in Figure 4.2 for continuous ADMM. We add these edges because, in practice, it takes no extra time to park a train at the end of the track rather than at the start.

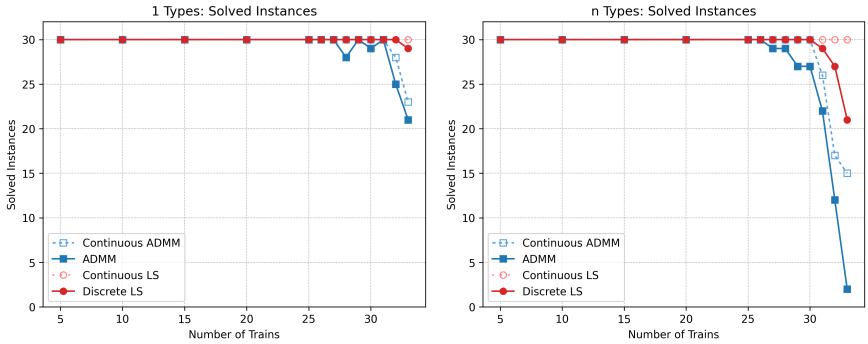
The cost of these edges is given by Equation 5.20 plus the LR and ALR penalties for each node that is skipped by this edge. So for the edge $(0, 4_3, t)$, the cost also includes the penalties for being at node 4_1 and 4_2 at time $t + 1$.

6.6.2 Experimental Evaluation

In this section, we discuss the results of the experiment done in Section 6.5.1 for continuous ADMM. We compare the performance of continuous ADMM with the performance of continuous LS and the ADMM approach without the extension. The performance is evaluated on the number of feasible solutions found and the average computation time.

Does continuous ADMM outperform ADMM/continuous LS in terms of feasible solutions found?

In Figures 6.10a and 6.10b, the number of feasible solutions found for ADMM, continuous ADMM, discrete LS, and continuous LS for 1 and n train types can be seen. We again only show these results as they are the outer train type values; the results for the other number of train types can be found in Appendix 9. The trend in the number of train types is again that, for more train types, fewer feasible solutions are found by continuous ADMM. For continuous ADMM, we observe that it finds significantly more solutions than ADMM, but still finds significantly fewer solutions than both LS versions. Out of the 1500 scenarios, continuous ADMM and ADMM were unable to find a solution for 86 and 175 scenarios, respectively. Discrete LS was unable to find a solution for 27 scenarios, whereas continuous LS found a feasible solution for all scenarios. From these results, we conclude that continuous ADMM has better performance than ADMM but is still significantly outperformed by both LS versions.



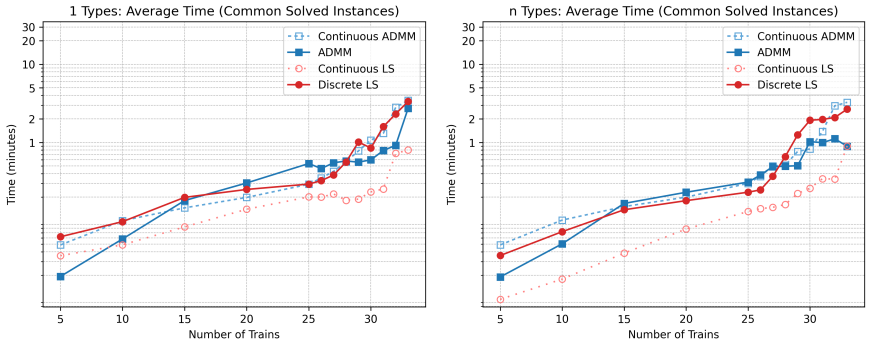
(a) (b)

Figure 6.10: Number of feasible solutions found for 1 and n types, for ADMM, continuous ADMM, discrete LS, and continuous LS.

Does continuous ADMM outperform continuous LS in terms of computation time?

In Figures 6.11a and 6.11b, the average computation time for ADMM, continuous ADMM, discrete LS, and continuous LS for 1 and n types can be found. To remove bias, these figures include only scenarios for which all approaches found a feasible solution. These figures use a logarithmic scale. The results for the other number of train types are again similar and can be found in Appendix 9. In this section, we focus on the comparison between continuous ADMM and continuous LS, as ADMM and discrete LS solve a different problem, and since the implementation of continuous ADMM has been improved in terms of computation time

compared to ADMM. This improvement was made since continuous ADMM has significantly more edges whose costs have to be set and computed. Setting and computing these costs require the most computation time. ADMM could be improved similarly, reducing its computation time; implementing this was outside of the scope of this thesis.



(a) (b)

Figure 6.11: Average computation time of commonly solved instances on a logarithmic scale for 1 and n train types for ADMM, continuous ADMM, discrete LS, and continuous LS.

In these figures, we see that the average computation time of continuous ADMM is significantly higher than that of continuous LS. From these results, we conclude that in terms of computation time, continuous ADMM performs worse than continuous LS. A computation time of at most four minutes is still a realistic time for practical use, though.

Chapter 7

Conclusions and Future Work

In this thesis, we aim to answer the following research question: **How well can Lagrangian Relaxation (LR) methods be applied to a simplified version of the Train Unit Shunting Problem (TUSP)?**

In the somewhat simplified version, different train lengths, the actual traversal times, and the (de)coupling of train units are left out. We contribute an Augmented Lagrangian Relaxation (ALR) approach solved with the Alternating Direction Method of Multipliers (ADMM) that has similar performance as the current state-of-the-art Local Search (LS) approach (**SQ2**, **SQ4**). The ADMM approach is significantly faster and finds significantly more feasible solutions than the LS approach on scenarios with smaller time windows and fewer feasible solutions (**SQ4**, **SQ5**). While for scenarios with a larger time window and with more trains, the LS approach is faster and finds significantly more feasible solutions than ADMM. On the scenarios for which ADMM does not find a feasible solution, it gets stuck in a cycle since the Lagrangian penalties are not high enough in later iterations and do not accumulate enough to explore different solutions (**SQ3**). One advantage of the ADMM approach is that it can allow for multiple trains to move at the same time, which is not possible in the LS approach.

We observe that the standard LR cannot find feasible solutions (**SQ2**) due to symmetry issues (**SQ3**). These issues are caused by symmetry in the trains and tracks. The MILP model shows that the routing, parking, and matching components of the TUSP can be modeled for LR with a time-extended graph in which tracks and switches are represented by nodes and edges, respectively (**SQ1**). The matching of trains is modeled through routing decisions, as only one train can depart at the same time, and thus match to the same train. The experimental evaluation demonstrates that the modeling decision to split tracks into multiple nodes has a significant effect on the complexity of the TUSP (**SQ1**). The continuous ADMM approach addresses this modeling choice by adding additional edges such that traversal of the edges between nodes originating from the same track is instant. The continu-

ous ADMM approach solves a more realistic version of the TUSP and finds more feasible solutions.

While the TUSP version solved in this thesis is simplified compared to the problem in practice, we believe that the ADMM approach is a solid alternative to the LS approach, as it performs significantly better for certain scenarios and addresses the main disadvantages of the LS approach. Namely, the LS approach is non-deterministic, and the planning of movements is a bottleneck, whereas the ADMM approach is deterministic and routing-focused. Furthermore, the ADMM's average computation time when finding a feasible solution is less than four minutes. This is a realistic time for use in practical planning operations, as it can quickly give a new solution when a change or disruption happens in the planning.

7.1 Future Work

The most important future work left after this thesis is extending the model with the parts of the TUSP that were left out. These parts are the actual length of trains, realistic traversal times, reversals, and (de)coupling. In future work, the actual length of trains could be modeled by adding capacity to tracks, and realistic traversal times could be included by giving edges different durations. The inclusion of reversals and (de)coupling into the model is less straightforward and would likely have to be solved with a pre- and/or post-processing step.

Aside from these parts, scheduling servicing tasks and planning personnel are also an integral part of the shunting problem in practice. These could be either included in the model or solved separately with other approaches. In practice, delays are allowed; modeling the possibility for delays could also be a useful extension.

As ADMM finds fewer feasible solutions than LS for scenarios with a larger time window and more trains, improving its performance for these scenarios is essential. When ADMM does not find a feasible solution, it often gets stuck in a cycle. Adding Tabu Search to ADMM or using different penalty update methods can help with escaping from these cycles. The two aspects of the subgradient method discussed in Section 6.4 also influence the performance of ADMM and could thus be improved in future work. Furthermore, primal recovery techniques could be used to construct feasible solutions from the relaxed solutions, as there are, on average, only 2 conflicts left when ADMM does not find a feasible solution. Another option could be to use the relaxed solution found by ADMM as the initial solution for the LS algorithm.

The experimental evaluation in this thesis is limited. ADMM and LS are evaluated for only a single shunting yard. Evaluating their performance across multiple yards is an important aspect for comparison and thus essential to research in future

work. Furthermore, the effect of the time window on the performance of ADMM and LS was only tested for 20 trains. Although it is expected that ADMM will also perform better for smaller time windows with more trains, we suggest that in future work, this is researched more thoroughly.

The number of movements is an essential factor for the feasibility of TUSP solutions that should be minimized. The current ADMM algorithm does not optimize for the number of movements. In the solutions found by ADMM, it often happens that trains have redundant movements and park unnecessarily in between movements. For future work, we propose to use an algorithm that reduces the number of movements after a feasible solution is found by ADMM instead of incorporating this into the model. As movement costs are already low, we expect that including this in the model will not have much effect.

Although the computation time of ADMM is realistic for use in practice, we believe that it could be sped up by only solving the set of trains that had conflicts every n iterations. A small experiment run showed that this could potentially increase the number of feasible solutions found and decrease computation time. As this was outside of the scope of this thesis, it was not researched thoroughly enough to include.

One downside of the LS approach that has not yet been mentioned is that it does not indicate whether a feasible solution actually exists. LR methods naturally lend themselves to generating lower bounds, which could potentially be used to determine if a scenario is feasible.

Overall, this thesis contributes an ADMM approach that can still be improved in numerous ways and is capable of solving a simplified version of the TUSP with similar performance to the current state-of-the-art LS approach.

Bibliography

C. Athmer. An evolutionary algorithm for the train unit shunting and servicing problem. Master's thesis, Delft University of Technology, 2021. URL https://resolver.tudelft.nl/uuid:0a65376e-bbb1-440b-95a2-586a369d9e98.

R. Bixby. *A brief history of linear and mixed-integer programming computation*, pages 107–121. 01 2012. ISBN 978-3-936609-58-5. doi: 10.4171/dms/6/16. URL https://doi.org/10.4171/dms/6/16.

Y. Bontekoning and H. Priemus. Breakthrough innovations in intermodal freight transport. *Transportation Planning and Technology*, 27(5):335–345, Oct. 2004. ISSN 0308-1060. doi: 10.1080/0308106042000273031. URL https://doi.org/10.1080/0308106042000273031.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011. URL https://doi.org/10.1561/2200000016.

M. A. Bragin. Survey on lagrangian relaxation for milp: Importance, challenges, historical review, recent advancements, and opportunities. *arXiv:2301.00573 [math.OC]*, 2023. URL https://arxiv.org/abs/2301.00573.

M. A. Bragin, P. B. Luh, J. H. Yan, N. Yu, and G. A. Stern. Convergence of the surrogate lagrangian relaxation method. *Journal of Optimization Theory and Applications*, 164(1):173–201, 2015. doi: 10.1007/s10957-014-0561-3. URL https://doi.org/10.1007/s10957-014-0561-3.

U. Brännlund, P. O. Lindberg, A. Nou, and J. E. Nilsson. Railway timetabling using lagrangian relaxation. *Transportation Science*, 32(4):358–369, 1998. URL https://doi.org/10.1287/trsc.32.4.358.

CBS. Hoeveel wordt er met het openbaar vervoer gereisd? URL https://www.cbs.nl/nl-nl/visualisaties/verkeer-en-vervoer/personen/openbaar-vervoer. 2026-04-30.

V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45 (1):41–51, 1985. URL https://doi.org/10.1007/BF00940812.

F. Clautiaux and I. Ljubić. Last fifty years of integer linear programming: A focus on recent practical advances. *European Journal of Operational Research*, 324 (3):707–731, 2025. doi: 10.1016/j.ejor.2024.11.018. URL https://doi.org/10.1016/j.ejor.2024.11.018.

Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 2). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2026.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981. doi: 10.1287/mnsc.27.1.1. URL https://doi.org/10.1287/mnsc.27.1.1.

M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981. URL https://doi.org/10.1002/net.3230110205.

R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman. Shunting of Passenger Train Units in a Railway Station. *Transportation Science*, 39(2):261–272, 2005. ISSN 0041-1655. URL https://www.jstor.org/stable/25769246.

M. Guignard. Lagrangean relaxation. *Top*, 11(2):151–200, 2003. URL https://doi.org/10.1007/BF02579036.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, version 12.0.0, 2024. URL https://www.gurobi.com.

J. T. Haahr, R. M. Lusby, and J. C. Wagenaar. Optimization methods for the train unit shunting problem. *European Journal of Operational Research*, 262(3):981–995, 2017. doi: 10.1016/j.ejor.2017.03.068. URL https://doi.org/10.1016/j.ejor.2017.03.068.

W. E. Hart, C. Laird, J.-P. Watson, and D. L. Woodruff. Pyomo—optimization modeling in python. *Springer Optimization and Its Applications*, 67:219–260, 2017. doi: 10.1007/978-3-319-58821-6_9. URL https://doi.org/10.1007/978-3-319-58821-6_9.

M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970. URL https://doi.org/10.1287/opre.18.6.1138.

J. N. Hooker. Integer programming: Lagrangian relaxation. In *Encyclopedia of Optimization*, pages 1–7. Springer, 2024.

J. E. Hopcroft and R. M. Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973. URL https://doi.org/10.1137/0202019.

IBM Quantum Developers. rustworkx: A high-performance graph library for python, 2023. URL https://github.com/Qiskit/rustworkx.

F. Kamenga. *Combinatorial optimization for integrating rolling stock management and railway track scheduling in passenger stations*. PhD thesis, Université de Lille, 2020. URL https://hal.science/tel-03169424v1.

M. H. Keaton. Designing optimal railroad operating plans: Lagrangian relaxation and heuristic approaches. In *Transportation Research Part B: Methodological*, volume 23B, pages 415–431. Pergamon Press, 1989. URL https://doi.org/10.1016/0191-2615(89)90042-8.

L. G. Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979. URL https://doi.org/10.1016/0041-5553(80)90061-0.

S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. URL https://doi.org/10.1126/science.220.4598.671.

N. Kohl and O. B. Madsen. An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations research*, 45(3):395–406, 1997. URL https://doi.org/10.1287/opre.45.3.395.

L. G. Kroon, R. M. Lentink, and A. Schrijver. Shunting of passenger train units: An integrated approach. *Transportation Science*, 42(4):436–449, 2008. doi: 10.1287/trsc.1080.0243. URL https://doi.org/10.1287/trsc.1080.0243.

R. Lentink, P.-J. Fioole, L. Kroon, and C. Woudt. *Applying Operations Research Techniques to Planning of Train Shunting*, pages 415 – 436. 02 2003. ISBN 9780471781264. doi: 10.1002/0471781266.ch15. URL https://doi.org/10.1002/0471781266.ch15.

R. M. Lentink. *Algorithmic Decision Support for Shunt Planning*. Phd thesis, Erasmus School of Economics, Erasmus University Rotterdam, Rotterdam, The Netherlands, 2006.

N. Lonyuk. Using pddl models to solve tuss: How to model tuss as an automated planning problem and solve it. Master's thesis,

Delft University of Technology, Delft, The Netherlands, December 2024. URL https://repository.tudelft.nl/file/File_f4f0af7f-22e0-4528-a550-dab9426a8fb1.

J. Mulderij, B. Huisman, D. Tönissen, K. van der Linden, and M. de Weerdt. Train unit shunting and servicing: a real-life application of multi-agent path finding. arXiv preprint, 2020. URL https://arxiv.org/abs/2006.10422.

G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

NS. Reizigersgedrag | NS Dashboard. URL https://dashboards.nsjaarverslag.nl/reizigersgedrag. Visited on 2026-04-29.

NVBS. Sprinter nieuwe generatie (sng). URL https://nvbs.com/kennis/railsystemen-nederland/spoorwegen-reizigersvervoer/nsr/sprinters/sng/. Visited on 2026-05-05.

OV in Nederland Wiki. Materieel. URL https://wiki.ovinnederland.nl/wiki/. Visited on 2026-05-05.

B. T. Polyak. Minimization of nonsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3):14–29, 1969. doi: 10.1016/0041-5553(69)90061-5. URL https://doi.org/10.1016/0041-5553(69)90061-5.

Python Software Foundation. Python language reference, version 3.10, 2021. URL https://www.python.org/.

Robust-Rail-NL. mathematical-models, 2026a. URL https://github.com/Robust-Rail-NL/mathematical-models.

Robust-Rail-NL. robust-rail-generator, 2026b. URL https://github.com/Robust-Rail-NL/robust-rail-generator.

Robust-Rail-NL. robust-rail-solver, 2026c. URL https://github.com/Robust-Rail-NL/robust-rail-solver.

G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*, pages 563–569, 2015. URL https://doi.org/10.1016/j.artint.2014.11.006.

H. D. Sherali and G.-J. Choi. Recovery of primal solutions when using subgradient optimization methods to solve lagrangian duals of linear programs. *Operations Research Letters*, 19(3):105–113, 1996. doi: 10.1016/0167-6377(96)00019-3. URL https://doi.org/10.1016/0167-6377(96)00019-3.

J. Shi, H. Li, H. Ruan, and P. Vansteenwegen. The shunting with service scheduling problem at a chinese high-speed railway depot. *Transportation Research Part C: Emerging Technologies*, 2023. URL https://lirias.kuleuven.be/retrieve/8b403c06-08a5-4eee-a3e6-e21802818276.

M. Song and L. Cheng. An augmented lagrangian relaxation method for the mean-standard deviation based vehicle routing problem. *Knowledge-Based Systems*, 247:108736, 2022. URL https://doi.org/10.1016/j.knosys.2022.108736.

D. A. van Cuilenborg. Train unit shunting problem, a multi-agent pathfinding approach. Master's thesis, Delft University of Technology, 2020. URL https://resolver.tudelft.nl/uuid:c9a0f41c-de6f-4ef4-8d94-6d0a89ef3eec.

R. W. van den Broek. Train shunting and service scheduling: an integrated local search approach. Master's thesis, Utrecht University, Utrecht, The Netherlands, 2016. URL https://studenttheses.uu.nl/handle/20.500.12932/24118.

R. W. van den Broek, H. Hoogeveen, M. van den Akker, and B. Huisman. A local search algorithm for train unit shunting with service scheduling. *Transportation Science*, 56(1):141–161, 2022. doi: 10.1287/trsc.2021.1090. URL https://doi.org/10.1287/trsc.2021.1090.

J. Wang et al. Synchronized optimization for service scheduling, train parking and routing at high-speed rail maintenance depot. In *IEEE Transactions on Intelligent Transportation Systems*, 2022. doi: 10.1109/TITS.2020.3045852. URL https://doi.org/10.1109/TITS.2020.3045852.

L. A. Wolsey. *Integer Programming*. John Wiley & Sons, New York, 1998. ISBN 9780471283669.

X. Xu and M. M. Dessouky. Train shunting with service scheduling in a high-speed railway depot. *Transportation Research Part C: Emerging Technologies*, 143:103819, 2022. doi: 10.1016/j.trc.2022.103819. URL https://doi.org/10.1016/j.trc.2022.103819.

W. Zhou and H. Teng. Simultaneous passenger train routing and timetabling using an efficient train-based lagrangian relaxation decomposition. *Transportation Research Part B: Methodological*, 94:409–439, 2016. doi: 10.1016/j.trb.2016.10.010. URL https://doi.org/10.1016/j.trb.2016.10.010.

Chapter 8

Appendix Conflicting Edge Sets

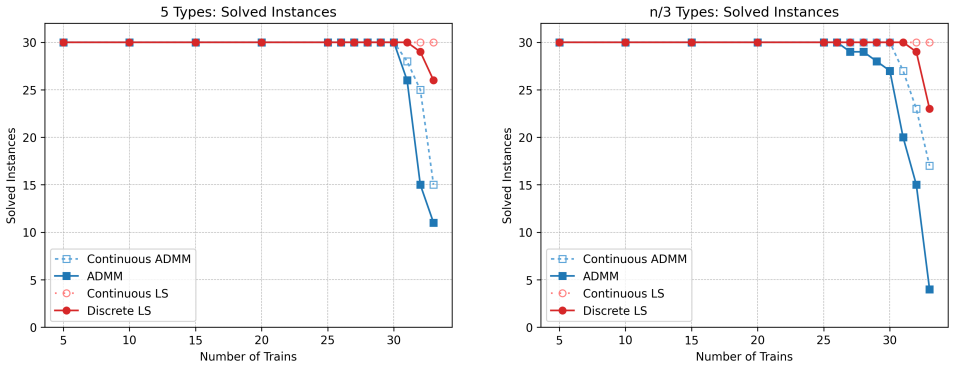
In Table 8.1, the conflicting edge sets for the Klein Binckhorst graph if multiple movements are allowed at the same time. The reverse edges are not included in this table, but each set also contains the reverse of each edge in that set. Furthermore, this table does not include edge sets for edges that only conflict with their reverse. For each of these edges, there is also a conflicting edge set, for example: $(1_1, 1_2)(1_2, 1_1)$.

Set #	Conflicting Edges
1	$(0, 1_1), (0, 2_1), (0, 3_1), (0, 4_1), (0, 5_1), (0, 6_1), (0, 7_1), (0, 8_1), (0, 9_1)$
2	$(2_4, 14_1), (2_4, 11_1), (3_4, 14_1), (3_4, 11_1), (4_3, 14_1), (4_3, 11_1),$ $(5_3, 14_1), (5_3, 11_1), (6_2, 14_1), (7_2, 14_1)(8_2, 14_1), (9_2, 14_1)$
3	$(2_4, 11_1), (3_4, 11_1), (4_3, 11_1), (5_3, 11_1), (6_2, 14_1), (6_2, 11_1),$ $(7_2, 14_1), (7_2, 11_1), (8_2, 14_1), (8_2, 11_1), (9_2, 14_1), (9_2, 11_1)$
4	$(6_2, 12_1), (6_2, 13_1), (7_2, 12_1), (7_2, 13_1), (8_2, 11_1), (8_2, 12_1),$ $(8_2, 13_1), (8_2, 14_1), (9_2, 11_1), (9_2, 12_1), (9_2, 13_1), (9_2, 14_1)$
5	$(6_2, 14_1), (6_2, 11_1), (6_2, 12_1), (6_2, 13_1), (7_2, 14_1), (7_2, 11_1),$ $(7_2, 12_1), (7_2, 13_1)$
6	$(8_2, 10), (8_2, 11_1), (8_2, 12_1), (8_2, 13_1), (8_2, 14_1)$
7	$(11_2, 15), (12_2, 15), (13_2, 15)$

Table 8.1: Overview of conflicting edge sets for the Kleine Binckhorst graph.

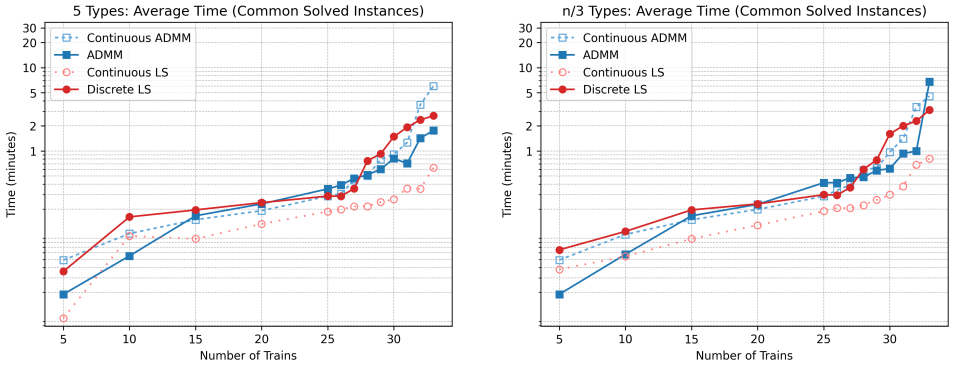
Chapter 9

Appendix Results



(a) (b)

Figure 9.1: Number of feasible solutions found for 5 and $n/3$ types for ADMM, continuous ADMM, discrete LS, and continuous LS.



(a) (b)

Figure 9.2: Average computation time of commonly solved instances on a logarithmic scale for 5 and $n/3$ train types for ADMM, continuous ADMM, discrete LS, and continuous LS.

Chapter 10

Appendix Generative AI Tools Usage

In this thesis, the following generative AI tools were used: ChatGPT, Grammarly, and GitHub Copilot. ChatGPT was used for the following tasks:

- Literature search.
- Creating citations.
- Creating LaTeX tables and LaTeX formatting.
- Assisting with mathematical notation.
- Code assistance: explaining code libraries (Pyomo), speeding up code, and answering syntax questions. Specifically, it was also used to assist in constructing the conflicting edges and to construct the additional edges for continuous ADMM.
- Data processing: creating code for running experiments, processing results data, and creating figures.
- Presentation support: tips for improving the layout of a few slides and creating one image.

Grammarly (basic version) was used for punctuation, word choice suggestions, and minor sentence adjustments, such as adding or removing words for clarity and grammar. GitHub Copilot was used for inline code and comments suggestions.