An Evolutionary Algorithm for the Train Unit Shunting and Servicing Problem

C. Athmer

Master of Science Thesis

TU Delft Delft University of Technology

Electrical Engineering, Mathematics & Computer Science (EEMCS)

An Evolutionary Algorithm for the Train Unit Shunting and Servicing Problem

by

C. Athmer

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on July 15, 2021.

Student number:	4329066	
Project duration:	July 1, 2020 – July 15, 2021	
Thesis committee:	Dr. M. M. de Weerdt,	TU Delft, supervisor
	Dr. ir. L. Bliek,	TU Eindhoven, supervisor
	Dr. A. Panichella,	TU Delft

TU Delft Delft University of Technology

Preface

This thesis is written as the final fulfillment of the requirements to obtain my master's degree in Computer Science at the Delft University of Technology. This report describes my research over the past 12 months on the development of an Evolutionary Algorithm for the Train Unit Shunting and Servicing Problem.

First of all, I would like to thank my supervisors, Mathijs de Weerdt and Laurens Bliek, for the time and effort they put in to guide me in my research. We met mostly on a weekly basis for 12 months, which was a big help for me to keep making progress, even at times when motivation was low. In particular, I want to thank you, Mathijs, for the interesting discussions we had about different parts of the algorithm, for example about different ways to do crossover. These were fun discussions and a big inspiration for the methods described in this thesis. And I want to thank you, Laurens, for the different perspectives you offered throughout my thesis. Also, I really appreciate you staying involved with my supervision after your move to Eindhoven University.

Research was done in collaboration with the Nederlandse Spoorwegen, for whom solving this problem efficiently is an important goal. I want to thank the Nederlandse Spoorwegen (NS), and in particular Bob Huisman, for giving me the opportunity to do the research for my master thesis as part of an internship at the NS. Additionally, I would like to thank all the colleagues of the R&D department for their interest and help throughout my research. In particular, I would like to thank Roel van den Broek, Chris Pustjens, Joris den Ouden, Jord Boeijink, and Demian de Ruijter for their help. Also, I would like to thank Leon van der Knaap who did his thesis at the NS during the same period, and the valuable discussions we had about our theses.

Finally, I want to thank my friends and family for their support during my thesis. It was a strange year with the corona pandemic, because of which I could not go to the university during the entirety of my thesis. Seeing friends and family throughout this year definitely helped to keep me sane during this project. In particular, I would like to thank my mother who has been a huge support during my thesis.

Casper Athmer
Delft, July 2021

Abstract

The Train Unit Shunting and Servicing (TUSS) problem is an NP-hard problem encountered by the Dutch railway operator, Nederlandse Spoorwegen (NS). It considers trains at a shunting yard when they are not transporting passengers. It consists of four subproblems, involving assigning the trains at the yard to a transporting timetable (*Matching*), planning the maintenance tasks that need to be performed for each train while they are at the yard (*Servicing*), assigning trains to parking tracks during their stay at the yard (*Parking*), and calculating routes for the movements of the trains over the yard (*Routing*). The problem entails finding a conflict-free plan that solves all subproblems.

Currently, plans for the trains at a shunting yard are still devised by human planners, which is a time-consuming and difficult task, which the NS wants to automate. To that end, they are in an advanced stage of the development of an algorithm that can solve instances of TUSS using a local search heuristic, called Hybrid Integrated Planner (HIP). Because it is a heuristic, it has no guarantee of being optimal. Another meta-heuristic that shows good performance on scheduling problems, is the class of Evolutionary Algorithms (EAs). This research creates an EA, called Conflict-Based Crossover (CBC), that can solve TUSS and can outperform the local search heuristic on a subset of locations.

The most complex part of creating an EA for a complicated problem like TUSS is devising a crossover operator, which is a method that can combine two solutions to TUSS to create a new, valid solution. This research devises a functional crossover operator for TUSS, using a graph representation of solutions which is also used in the local search heuristic. Using the same representation as HIP makes it easy to hybridise the algorithm. The operator treats schedules for individual train units as variables and combines solutions by mixing these variables. To decide which variables are selected from which solution, a heuristic is developed that uses the conflicts of individual train units as the basis of a selection mechanism; the method derives its name from this heuristic.

The performance of CBC is compared to HIP on a set of different shunting yards. Results show that on Grote Binckhorst, which is a location where most tracks have only one entry and exit, CBC gets outperformed by HIP. However, on Kleine Binckhorst, a location with a lot of open-ended tracks, CBC consistently outperforms HIP. This implies that, on a subset of locations, CBC can be an improvement over HIP to devise feasible plans for TUSS.

Table of Contents

Chapter 1

Introduction

The Dutch rail network is the third busiest in the world after Switzerland and Japan [1] indicating that public transport by train plays an important role in Dutch society, and its demand is ever increasing. The Nederlandse Spoorwegen (NS) is the main railroad operator in the Netherlands, and the number of passengers transported per day increased from 1.1 million in 2015 to 1.3 million passengers a day in 2019 [2]. The climate goals within the European Union drive countries to stimulate consumers to travel by train rather than by plane within Europe. Within the Netherlands, parking prices and other regulations discourage people from travelling to cities by car, encouraging them to use public transport. These developments further increase the demand for transport by train. Because of this, the NS is in the process of increasing their rolling stock (i.e. the number of trains that are available to them for transporting passengers) from 2966 trains in 2017 to 4077 trains in 2022, an increase of 37% [3].

This increase in trains on the rail network poses problems for the NS regarding shunting (i.e. parking) and servicing (i.e. maintenance, cleaning, etc.) trains. When a train is not transporting passengers it is parked at a shunting yard, where it is also being serviced. Most of the trains are at the shunting yard during the same period, which is during the night when there are very few trains transporting passengers. These shunting yards have limited capacity. They are usually located close to large train stations and there is not always space to expand them (which is also very expensive). Therefore it is important to develop efficient plans regarding the parking and servicing of trains on shunting yards, to ensure that as many trains can be sent to a particular shunting yard as possible. The problem of designing these plans is called the Train Unit Shunting and Servicing (TUSS) problem, which is an NP-hard feasibility problem for which no exact solver exists that can solve real-world problem instances.

TUSS is made up of four subproblems. The first one is *matching*, which is the problem of assigning incoming trains to departures the next morning. The second problem is called *servicing*, which entails scheduling all service tasks that need to be performed for each train before that train departs from the shunting yard. The third subproblem is the *parking* problem, which involves finding a track for each train to park on when it is not being serviced

while ensuring that the capacity of a track is not exceeded, since tracks have limited length. Finally, a train will have several movements across the shunting yard (e.g. moving from its parking track to a service resource) and unobstructed routes need to be calculated for each movement, called the *routing* problem. If all constraints from every subproblem are satisfied, a feasible plan emerges which can be used in practice.

Currently, these plans are made by human planners on a daily basis, for a 24-hour horizon. A problem instance can entail more than 50 trains that are at the shunting yard at a single point in time and can be a difficult puzzle for a human to solve. To this end, automated plan generation could aid planners in devising a feasible plan. Another use case of automated plan generation is to determine the maximum capacity of shunting yards, which can guide the NS in strategical decision making regarding new or existing shunting yards (e.g. whether a shunting yard should be expanded).

The NS is currently developing a local search method to generate these plans automatically and aims to put it into use by 2023 [4]. The method should at least perform at an equal level as human planners (i.e. it should at least always find a solution when a human planner finds one), and preferably outperform humans. However, experiments show that the local search method does not always find a solution when it does exist, or that it only finds a solution if the right starting position is chosen. This implies that there is room for improvement, because an ideal scenario would be that an algorithm always finds a solution when it exists. Since the problem is too complicated for exact methods, this cannot be guaranteed. But if an algorithm can be devised that can solve more problem instances than the local search method of the NS, it is closer to this ideal scenario.

A possible class of algorithms that may be able to outperform local search is the class of Evolutionary Algorithms (EAs). Like local search, an EA is a heuristic method, which means it is fast. EAs have shown success with problems that have to do with job scheduling [5, 6, 7, 8]. Since TUSS is also a job scheduling problem at its core, an EA may also work well for TUSS. To this end, the aim of this research is to investigate whether an evolutionary algorithm can be developed that can solve TUSS, and whether this method can outperform the local search method currently in use.

From a research perspective, it is interesting to assess whether TUSS can be solved with an EA. If so, it shows a practical application of an EA, solving a hard, real-world feasibility problem. Furthermore, (part of) the method devised for TUSS can possibly be ported to other (scheduling) problems.

This chapter continues in Section 1-1 with a discussion of related work regarding TUSS. This gives a comprehensive overview of what has already been done, and it shows what is lacking in the literature. This research can potentially address some of these gaps in the literature. Section 1-2 goes into more detail about the contents of this research: it lays out and elaborates on the research questions that are being answered, it sums up the contributions, and it gives an overview of the contents of each chapter in the remainder of the thesis.

1-1 Related work

Since TUSS is an important issue for railroad operators, (part of) the problem has already been researched in the past. This section describes the most important literature regarding

TUSS and its variants.

The Train Unit Shunting Problem (TUSP) was introduced by Freling et al. in 2005 [9]. In their work only matching and parking are considered, i.e. service scheduling and routing were not part of the problem formulation. The authors use a (mathematical) decomposition approach to solve the problem. This work has been expanded in the years after publication with similar approaches [10, 11, 12], but the mathematical formulations take long to solve and become too complex to solve for large instances.

Jacobsen and Pisinger [13] present three heuristic approaches for TUSP based on, respectively, Guided Local Search, Guided Fast Local Search, and Simulated Annealing. The authors did their research for the Danish railroad network, which has a similar structure to the Dutch railroad network.

Van den Broek [14] developed a heuristic solution to TUSS specifically for the Dutch railroad, which is currently being refined and tested by the NS, called Hybrid Integrated Planner (HIP). It uses simulated annealing to find a feasible solution to instances of TUSS. To apply local search, heuristics are used to create an initial solution after which local search operators are applied to the initial solution to try and improve it.

Haahr et al. [15] compare multiple solution approaches, including a constraint programming formulation, a column generation approach, and a randomized greedy heuristic. These approaches are compared with two existing methods, a mixed-integer linear program and a two-stage heuristic. The authors conclude that the column generation method is outperformed by all other approaches and that exact mathematical models quickly become too large, consuming a lot of memory. The heuristic methods performed best and were able to solve most feasible instances they were tested on.

Peer et al. [16] formulate the problem as a Markov Decision Process (MDP) and develop a Deep Reinforcement Solution. This approach allows for online planning, which is useful under uncertainty. A drawback of the method is that it finds feasible solutions for significantly fewer instances than the aforementioned local search method in [14].

Van Cuilenborg [17] attempts to solve TUSP by modelling it as a Multi-Agent Pathfinding (MAPF) problem. The emphasis of this research is on routing, since in some cases heuristics are not able to find a detour route when the initial route of a train is blocked by another train. It solves this MAPF model using different variations of Conflict-Based Search. However, similar to the deep reinforcement research mentioned previously, this research finds fewer feasible solutions than the state-of-the-art local search method developed by the NS.

In the literature, one effort can be found that uses an EA for (part of) TUSS. Jekkers [18] developed two genetic algorithms for the problem, focusing on parking and matching of train units, leaving resource scheduling and routing out of the scope of the research. The first algorithm has four decision variables or 'genes' for each individual which are being optimized. The genes describe which train units are parked at each location and the arrival times of the train units, as well as the departure times and a matching of parked train units to departing trains. In the second algorithm, a greedy heuristic is used for the matching to minimize the number of crossings on departure. This approach is currently used to aid the construction of shunt plans for shunt yards located near *Rotterdam Central Station* and *Hoofddorp*, two major stations operated by NS [14].

The literature from the aforementioned works shows that exact methods are currently not a viable option because of the complexity of the problem, and no exact method exists that can solve problem instances of realistic sizes. Heuristic methods are more successful, and most use some variation of local search to iteratively adapt an existing solution with small changes. Advantages of these local search methods are that they run relatively fast and the way they operate is intuitive, making it easier to understand them, opposed to mathematical models. A disadvantage is that the success of an individual run of such a local search heuristic is highly dependent on the starting point of the search, which is usually an initial solution created using heuristics. If this initial solution is 'far away' from a feasible solution in the search space, it is unlikely that local search will find that feasible solution. This is likely the biggest caveat of these methods and where the most progress can be made.

The success of evolutionary algorithms is generally less dependent on initial solutions because they make bigger changes to existing solutions, making it easier to 'jump' from an unfavourable starting point to a more promising area of the search space. An EA could therefore possibly be an improvement over the current state-of-the-art local search methods. Another advantage of EAs may be that it allows a human planner to choose from a variety of different solutions, because there is a population of individuals. This can be useful because some plans might be feasible, but not desirable in practice (e.g. because they require a lot of manpower to execute the plan). There is little work on EAs for TUSS, and the work that does exist does not solve the complete problem. This provides an opportunity to create an algorithm for TUSS that can outperform state-of-the-art methods.

1-2 Thesis contents

The previous section described previous efforts to solve problems related to train unit shunting and servicing. Just one paper exists that implements an EA to solve the parking and matching problem. It does not include servicing or routing, however, nor does it propose methods to include these in the method. This makes it incomplete, thus no conclusions can be drawn from it regarding EAs for TUSS. The aim of this research is to develop an EA that can solve TUSS in its entirety and that can outperform existing methods, leading to the main research question:

Can an evolutionary algorithm be developed that is able to solve more TUSS problem instances than the current state-of-the-art local search heuristic?

Four sub-questions are used to answer the main research question, which are presented below. In a complicated problem such as TUSS, the crossover operator, which specifies how two different solutions can be combined to form a new solution, is arguably the most important aspect of an EA. To be able to solve TUSS effectively using an EA, a well-designed crossover operator is important. That leads to the following research question:

RQ 1. What is a useful crossover operator for TUSS?

What is most relevant to the NS is the performance of the EA compared to HIP. This can be broadly split into two categories: using the algorithm to aid or replace human planners when making day-to-day plans, or to determine the maximum capacity of shunting yards to assist in strategic decision making. The former requires the algorithm to find solutions within a relatively small time window. The latter puts the focus on finding feasible solutions for

instances with as many trains as possible, without a constraint on the time it takes (within reason). This leads to the following research questions:

RQ 2. Given a limited budget, does the EA solve more problem instances than HIP?

RQ 3. Given an unlimited budget, does the EA solve more problem instances than HIP?

Finally, it is valuable to know in which scenarios the EA does or does not perform better than the local search heuristic. And even more important is the answer to the question why it does or does not outperform the local search method. These insights can help to determine future research directions for TUSS, and they may give an indication to know in which cases an EA can be used for TUSS. This leads to the final research question:

RQ 4. When and why does an EA (not) outperform local search for TUSS?

This paper has several contributions that add to the existing literature on TUSS and EAs. Most importantly, it presents an evolutionary algorithm that can solve TUSS and that can outperform the current state-of-the-art local search heuristic on a real-world location with problem instances that have a close resemblance to reality. For the algorithm to function, a crossover operator is designed that allows for mixing two different solutions to TUSS together to form a new solution. Furthermore, a heuristic is developed within this crossover operator which decides which parts from each solution to transfer to the new solution when combining multiple solutions to form a new solution. Finally, a string representation for TUSS is introduced which is used as the basis for a diversity measure between solutions, which is an indicator of the similarity of different solutions. This diversity measure is also the basis of a heuristic for deciding which solutions to combine.

The remainder of this thesis is as follows. Chapter 2 describes literature on heuristic methods, focusing on local search, evolutionary methods and hybrid solutions. After a review of the literature, the local search method currently employed by the NS is discussed in Chapter 3 together with an extensive problem description. This local search method is used as a benchmark and the solution representation that is used in HIP forms the basis of the evolutionary method developed in this thesis, which is laid out in Chapter 4. To evaluate the developed method, several experiments are run. Chapter 5 displays and discusses these results. Finally, Chapter 6 concludes the thesis and points to possible future research.

Chapter 2

Heuristic methods for Train Unit Shunting and Servicing

The literature on Train Unit Shunting and Servicing (TUSS) in Section 1-1 shows that the problem is too complex to solve using exact algorithms. An exact algorithm is guaranteed to solve a problem to optimality or feasibility (depending on the type of problem), given enough time. For TUSS, the most successful solutions developed to date use heuristic methods. Heuristics are methods to find good (near-)optimal solutions with a reasonable computational cost, without guaranteeing feasibility or optimality [19]. Meta-heuristics are a set of intelligent strategies to enhance the efficiency of heuristic procedures [20]. In general, many of the best performing techniques to solve complex real-world problems are meta-heuristics [21]. Examples of meta-heuristics are Ant Colony Optimisation (ACO) [22], Evolution Strategies (ES) [23], Particle Swarm Optimization (PSO) [24], Simulated Annealing (SA) [25], Tabu Search (TS) [26], and Variable Neighbourhood Search (VNS) [27]. These techniques differ in the search pattern applied to explore the search space, but they are all aimed at providing a balance between diversification (or exploration of the search space) and intensification (or exploitation of already found solutions) [21].

Arguably the most successful attempt at solving TUSS is the method developed by van den Broek [14], which is briefly described in Section 1-1. This is a meta-heuristic method that uses VNS, a stochastic local search method that is part of the class of meta-heuristic methods, to find a feasible solution to TUSS. This local search method is called Hybrid Integrated Planner (HIP) and is currently being refined and further developed at NS, to be used in practice.[1]

Another meta-heuristic method that has many successful applications is the class of Evolutionary Algorithms (EAs). As can be seen from the previous literature on TUSS described in Section 1-1, EAs for TUSS have not been sufficiently studied. The main goal of this research is to fill this void and determine if an Evolutionary Algorithm can be developed that outperforms

[1] In the literature, HIP is described as a Simulated Annealing method. Since then, it has been modified to use Variable Neighbourhood Search instead, because this yields better results

HIP, the state-of-the-art local search method described in the previous paragraph. The following section presents necessary background information for these methods and discusses the (dis)advantages of the respective methods. Furthermore, a hybrid of an EA and local search is discussed.

2-1 Local search and evolutionary algorithms

Generally speaking, local search algorithms start at some initial search position and iteratively move, based on local information, from the current position to neighbouring positions in the search space. Search methods can be split into two categories, viz., perturbative search and constructive search [28]. The former starts from a complete candidate solution and changes that solution to obtain a new valid solution. The latter starts from an "empty" candidate solution and iteratively extends it until a complete candidate solution is obtained. Most local search algorithms use randomised decisions for generating the initial solution or when determining search steps; these methods are referred to as Stochastic Local Search (SLS) [29]. The search process in any local search is usually guided by an evaluation function, which assigns a numerical value to the quality of a solution. Examples of SLS methods are Simulated Annealing [25], Tabu Search [26], Iterated Local Search [30] and Variable Neighbourhood Search [27].

EAs were introduced by Holland [31] and mimic the biological process of evolution. An EA is initialized with a *population* of *individuals*, which represent solutions to the problem that is being solved [32]. An EA runs over multiple *generations* until a stop criterion is met, where each generation consists of a (different) population which is created based on the population of the previous generation. *Variation* and *selection* are used to create a new population, based on the current one. Variation is the process of creating new individuals, i.e. *offspring*. In Genetic Algorithms (GAs), the most well-established method in the class of EAs, this is done using *mutation* (i.e. changing part of an individual) and *crossover* (i.e. combining two or more individuals to create a new individual) [32]. *Parent selection* determines which subsets of individuals from the current population are selected for crossover. However, there are other types of EAs that use different variation procedures. One example is a type of EA called Estimation of Distribution Algorithm (EDA), which creates a new population by sampling from an explicit probabilistic model constructed from promising solutions found so far [33]. Selection is the process of determining *survivors* that go into the next generation. A *fitness function* assigns a heuristic numerical estimate of quality to members of the population to guide the selection process [34]. Figure 2-1 shows a general overview of the flow of an EA. The next section details advantages and disadvantages of EAs and local search methods, and outlines literature comparing the respective methods.

2-1-1 Comparison of local search and evolutionary algorithms

Numerous advantages of using EAs are mentioned in the literature [36, 37, 38]:

- It is conceptually simple, making it relatively easy to understand and implement.
- It has a broad applicability, making it suitable for a large range of problems.

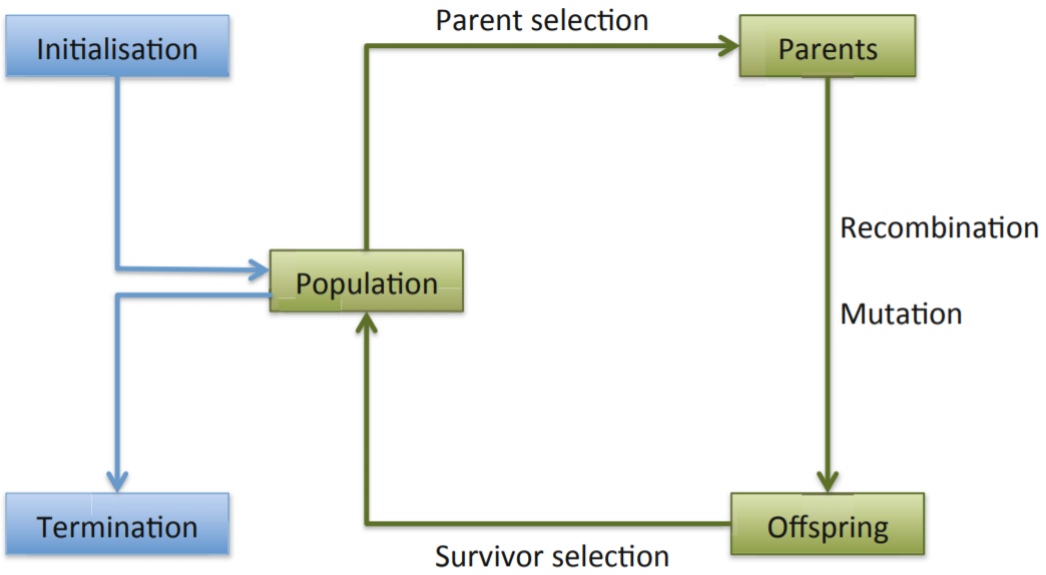

Figure 2-1: The general scheme of an evolutionary algorithm as a flowchart (taken from [35])

- It outperforms classic methods on real problems.
- It is possible to use problem knowledge and to hybridize with other methods.
- It allows for parallelism.
- It is robust to dynamic changes.
- It has the capability for self-optimization (meaning little parameter tuning).
- It is able to solve problems that have no known solutions.

Many, albeit not all, of the aforementioned advantages also apply to local search methods. The one advantage that is not directly applicable to local search is parallelism. However, local search methods can take advantage of parallelism by starting the algorithm with several different initial solutions, and running local search on these different solutions in parallel.

One advantage that is specific to an EA, is that diversity can be forced in the population [39], which means there can be multiple solutions with different characteristics. This can be particularly useful for TUSS. When a plan is found in advance and something changes (e.g. a train is delayed) which renders the chosen plan infeasible, a different feasible solution found by the algorithm might still be feasible given the new circumstances. Another advantage of this diversity characteristic is that NS can choose from multiple feasible solutions, allowing them to choose the plan that best fits their additional preferences.

A key difference between local search and EAs is that the latter use some sort of variation based on the population. For a GA, variation consists of mutation and crossover. Doerr et al. [40] found that crossover has a positive impact on the convergence speed of the search compared to a GA that does not use crossover (which is identical to local search with multiple

starts), implying that a proper use of variation can give EAs an advantage over local search methods.

Generally speaking, EAs cover a larger part of the search space than local search methods, because they have a population and use variation, whereas local search only changes the existing solution, and will thus make smaller steps in the search space. Therefore, local search is more likely to be trapped in a part of the search space where no good solution can be found.

However, local search does try to improve an existing solution and will therefore (eventually) always end up at a (local) optimum. The same cannot be said for EAs since variation puts the search in a different part of the search space and mutation does not consider whether it improves a solution.

A disadvantage of EAs can be that for some problems a large population size is needed for convergence, thereby increasing the computational time needed to find a solution [41]. Local search will generally converge fast, although this can mean that the algorithm is stuck in a (bad) local optimum. A workaround to this is to restart the search multiple times from different initial solutions.

Furthermore, there are many problems where suitable local search algorithms can find the global optimum. Because local search is generally easier to implement and faster than an EA, it is preferred in these cases [41]. An example of this is the Traveling Salesman Problem (TSP), where a local search method is currently considered the best performing algorithm and can solve many TSP instances to optimality [42]. It should be noted that HIP does not find a feasible solution for every instance of TUSS where a feasible solution exists, indicating it can be improved.

Finally, there is a copious amount of literature comparing the performance of EAs and local search methods for different problems. For certain problems EAs perform better [43, 44], for other problems local search performs better [45, 46, 47], and finally there are problems for which it depends on preferences or problem instances [48, 49]. No other conclusion can be drawn from this than the insight that it depends on the problem whether an EA or local search method is the best choice, which is in line with the No Free Lunch theorem [50].

2-1-2 Memetic algorithms

The previous section shows that both EAs and local search have their merits and drawbacks and that one cannot be preferred over another in the general case. A logical follow-up to this is to use a hybrid method that incorporates the advantages of both EAs and local search. Such hybrid methods are called Memetic Algorithms (MAs) and have gained a lot of popularity in recent years. Neri and Cotta [51] define a MA as follows:

> Memetic algorithms are population-based mechanisms composed of an evolutionary framework and a set of local search algorithms which are activated within the generation cycle of the external framework.

The structure of a MA is similar to that of an EA. The main difference is that local search is applied to every individual in the generated offspring, in an attempt to improve its fitness

[51]. A different way to look at it is that each individual is steered towards a local optimum with a local search method, and therefore the EA acts only on local optima.

Many problems have been tackled with this hybrid approach, and it has been shown to be able to consistently outperform pure local search [41]. Hybridization provides a good balance between exploration and exploitation of the search space, thereby decreasing the risk of getting stuck in a local optimum or not converging to any optimum.

Based on the literature on TUSS and heuristic methods discussed, it is expected that a MA can achieve state-of-the-art performance on TUSS. A local search method (i.e. HIP) is already being developed by NS and achieves good results, however, room for improvement remains. Since a well-functioning local search algorithm already exists, the focus of this research is mostly on developing an evolutionary algorithm, which can be combined with (an adaptation of) HIP. The explanation of EAs in Section 2-1 is a high-level explanation of the general workings of an EA. However, within this class of algorithms many variations exist. The next section describes the general scheme of an EA and the different choices that can be made for the various components within this scheme, and a variation on this scheme. Both are tried and tested in this research.

2-2 Components of Evolutionary Algorithms

Since the introduction of the Genetic Algorithm in 1975 [31], many different EAs have been developed for a wide array of problems. Although the implementation and the details across these algorithms widely varies, they all share the general scheme that is depicted in Figure 2-1. This section aims to explain the different components of this scheme, and outlines different choices that can be made for each component. The information in this section is based on the book by Eiben and Smith [35], which provides a detailed overview of evolutionary computing.

Finally, a more specific scheme, called Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA), is discussed. GOMEA makes specific choices for various components in the scheme.

The information in this section provides the basis for the development of Conflict-Based Crossover (CBC), the evolutionary algorithm presented in this thesis. In this research, both a custom setup (where each component of an EA is tuned based on domain knowledge and preliminary experiments) and GOMEA are tried and tested for TUSS.

2-2-1 Representation

The development of an EA starts by finding a representation of solutions to the problem that captures all important aspects of the problem. This is a vital part of the algorithm and also one of the most difficult parts of designing a good EA. Often, a good understanding of the application domain is needed to come up with a proper representation. The variation operators that are discussed in Section 2-2-5 depend on the representation that is used, and this representation plays a critical role in the success of an EA [52].

In EAs, *phenotypes* represent possible solutions to the problem at hand. In the case of TUSS, this is a schedule that details all actions of all trains from the moment they arrive at the shunting yard until the moment they depart.

A *genotype*, on the other hand, is a mapping from a phenotype to a representation that is used within the algorithm. For example, if there is an optimisation problem for which the solution is an integer, that is the phenotype. And a genotype could then be a binary representation of that integer, which is used within the algorithm.

For TUSS a graph representation of the schedule is used within the algorithm. The details of this representation are discussed in Section 3-2-2.

2-2-2 Evaluation function

The *evaluation function* or *fitness function* of an EA determines the quality of a solution and is equivalent to an objective function for other optimisation methods. The goal of the algorithm is to optimise (usually minimise) this value.

It is most common to calculate it using the phenotype representation since that is the direct representation of the real world. An issue with TUSS is that it is a feasibility problem (opposed to an optimisation problem), which makes it non-trivial to determine the quality of solutions. Section 2-3 discusses known ways to address this.

2-2-3 Population

The *population* in an EA is a set of *individuals*, i.e. genotypes, that changes over time by adding and removing individuals. Defining a population can be as simple as specifying the number of individuals in it. More elaborate methods exist (e.g. adding a spatial structure) but are not considered in this research. The population size is usually constant, and often trial and error is used to find the population size that performs best.

When an EA starts, a set of individuals need to be created to start the first generation with. This is the *initialisation* phase. Depending on the problem, solutions can be either randomly generated, or domain knowledge can be used to create problem-specific heuristics to create initial solutions. The latter method is used in this research.

Diversity is a measure of the spread of the individuals in the population. The way to measure diversity is problem-dependent. A common hypothesis within the literature on EAs is that a high diversity is important to avoid premature convergence and to escape local optima [53]. Using this insight, this research creates a diversity measure for TUSS that is both used for analysis and to guide the EA, as is explained in Section 4-2.

2-2-4 Parent selection

The goal of *parent selection* is to select individuals from the current population that will become parents of the next generation. It is usually probabilistic, and a common way to select parents is by correlating the quality of a solution to the chance that it gets picked as a parent.

2-2-5 Variation operators

The goal of variation operators is to create new individuals from old ones (i.e. from the parents selected during the parent selection phase). Two different types of variation exist: *mutation*, which is a unary operator, and *recombination*, or *crossover*, which is an n-ary operator.

Mutation is a stochastic operator, that makes a random, unbiased change to a genotype with a certain probability. In most EAs, it is more of a 'background' operator.

Crossover is the defining part of an EA. It merges information from two parents into one or two new genotypes, called offspring. It can be a stochastic operator but does not need to be.

Mutation and crossover operators are representation dependent. The crossover operators developed for the EA for TUSS are discussed in Section 4-4.

2-2-6 Offspring selection

Offspring selection or *survivor selection* determines which individuals survive into the next generation. Since the population size is usually constant, and variation creates new individuals, it means that some individuals from the set of original individuals extended with the new individuals, need to be discarded. The fitness value of the individuals is used to determine which individuals survive. However, usually something more sophisticated is done than simply taking the best n individuals (given a population size of n), to retain diversity in the next generation and prevent premature convergence. The method used in this research is *tournament selection*, which is detailed in Section 4-1-3.

2-2-7 Termination criteria

EAs are iterative algorithms, and at the end of every generation a check is made whether the algorithm should continue onto the next generation, or if it should terminate. Ideally, it terminates when an optimal (or feasible) solution has been found. However, in real-world scenarios the optimum is usually not known, and since an EA is a heuristic algorithm it is also not guaranteed to find the optimum. To prevent it from running indefinitely, other termination criteria can be defined. Common ones are:

- The maximally allowed CPU time has elapsed.
- The number of fitness evaluations passes a predefined threshold.
- The improvement of the best fitness value remains under a predefined threshold value for a set period of time (e.g. a couple of generations).
- The population diversity drops under a certain level.

All of these are valid termination criteria that can be used exclusively or in combination. It depends on the purpose of the experiments which criteria fit best.

2-2-8 Hybridisation

Section 2-1-2 discusses the benefits of a hybrid form of an EA and local search, called a Memetic Algorithm (MA). The most intuitive way to hybridise these methods is to add a local search heuristic (e.g. a hill-climbing heuristic) after the variation phase. The intuition behind this is that every individual is steered towards a local optimum, and crossover is only applied between local optima. This is also how hybridisation is applied in this research.

2-2-9 GOMEA

Finally, we look at a recently developed EA called GOMEA [54]. Its basics are the same as any EA, but it has some specific characteristics. It has achieved positive results on real-world applications in recent years [55, 56, 57]. One of the aforementioned papers applies GOMEA to the permutation flowshop scheduling problem [55]. A part of solving TUSS also involves permutations, which is why principles from this paper and GOMEA in general might apply well to TUSS. Another reason why it is hypothesised that this algorithm can work well for TUSS is that GOMEA groups variables that make up the individuals, such that variables that are dependent on each other are grouped together. Since we suspect that parts of the solution of TUSS are highly dependent on each other, GOMEA can group these parts together and have a positive impact on the results of the EA.

The groups of variables that are used in the algorithm are called a Family Of Subsets (FOS). If you have a set of variables S, a FOS is a subset of the powerset of S. Often, every variable is in at least one FOS subset; this is called a linkage set. An example is the univariate FOS which is defined by putting every variable in its own subset; this means that each variable is modelled to be independent of all other variables. Many automated methods to decide the grouping of the variables exist, however it is outside of the scope of this research to discuss these here. Because there is a lot of available information about TUSS and its problem instances, it is hypothesised that the best way of making groups is to use domain knowledge.

The other defining characteristic of GOMEA is the use of intermediate function evaluations inside the variation stage. Using these intermediate evaluations, GOMEA greedily improves solutions. It works as follows: for every individual, loop through all the subsets in the FOS. For each subset, pick a random donor solution (which is another individual from the population). Perform crossover by selecting the values for the variables defined in the current subset from the donor solution, and inserting these values in the receiving solution. At this point, local search can be applied in the case of hybridisation. Afterwards, check if the new solution has an improved fitness value. If so, replace the original with the new solution, otherwise discard it. Because new solutions automatically replace the original when an improvement is made, elitism is guaranteed and no offspring selection phase is necessary. Algorithm 1 shows pseudocode for GOMEA, in line with the explanation given above.

An important note is that most of the literature on EAs and local search focuses on optimisation problems, which aim to find the best solution among the set of all feasible solutions. TUSS however, is about finding any feasible solution, which poses some issues compared to optimisation problems. The next section discusses the application of EAs to this class of problems.

Algorithm 1 Pseudocode of GOMEA

```
F ← Create FOS
P ← Initialise population of size n
terminate = false
while !terminate do
    for each individual i ∈ P do
        f_o = EvaluateFitness(i)                          ▷ Evaluate original fitness
        for each subset F_j ∈ F do
            d = Random(P_0, P_1, ..., P_{n-1})            ▷ Get a random donor
            i_{F_j} = d_{F_j}                             ▷ Crossover for variables in current subset
            f_n = EvaluateFitness(i)                      ▷ Evaluate fitness after crossover
            if f_n > f_o then
                Keep changes made by crossover
            else
                Revert changes made by crossover
    if termination criteria are met then
        terminate = true
```

2-3 Constraint satisfaction problems

In the previous sections, benefits and drawbacks of local search methods, EAs and MAs, which are described in the literature, were discussed. The general layout of an EA was also discussed. Most of the literature on heuristic methods focuses on applying these methods to optimisation problems. However, TUSS is a feasibility problem, meaning that the goal is to find any feasible solution that satisfies all the constraints. Problems of this type are also called Constraint Satisfaction Problems (CSPs).

Any standard EA does not take constraints into account, since the regular search operators are 'blind' to constraints [58]. This means that even if the parents are satisfying all the constraints, there is no guarantee that the offspring will also satisfy all constraints. This suggests that EAs are intrinsically not well-suited for CSPs [59]. However, there are techniques that allow for the use of EAs with a CSP and have been proven to be effective in practice, which are discussed in the remainder of this section.

One way to deal with constraints when using an EA is to handle them directly. With direct constraint handling the EA is modified to ensure that all constraints are enforced at all times [59]. Note that for a CSP this is not possible, since the goal is to find a solution that satisfies all constraints, so enforcing this creates a new CSP. However, direct constraint handling can be used for part of the constraints (the other constraints will then be handled indirectly), which is why direct constraint handling techniques are also relevant for CSPs and discussed here.

Typical approaches to handle constraints directly are the following [59]:

- Eliminating infeasible candidates. Since the goal is to find a feasible candidate, it is unlikely that feasible offspring is created right off the bat in a CSP (because then the algorithm could terminate anyway). Therefore, this is not a useful technique here.

- Repairing infeasible candidates, which incorporates a procedure to transform infeasible solutions into feasible ones [60].
- Preserving feasibility by special operators, which means designing a special representation and operator such that variation in the population always keeps the solutions feasible [60]. An example of this can be found in the EA developed by Jekkers [18], which uses cyclical crossover to always keep a valid matching.
- Decoding, which shifts the search space to a space where *free search* can be applied. The elements of the new search space serve as inputs for a decoding procedure that creates feasible solutions [59].

Because the aforementioned approaches for direct constraint handling are, per definition, not applicable to an entire CSP, an (additional) indirect constraint handling approach is needed. One method is to use penalty functions: this incorporates the violation of constraints in an objective function by assigning a penalty value to a violation of each constraint [60]. When the objective function is zero, all constraints are satisfied and a feasible solution has been reached. This technique is used in HIP and the EA developed in this research. Other indirect constraint handling techniques exist but are omitted here for the sake of brevity.

To summarise, constraint handling techniques are needed to solve CSPs with an EA. Direct constraint handling techniques ensure that constraints are always met. These cannot be used as standalone methods because the goal is to find a solution that meets all constraints in the first place. However, they can be used for part of the problem, and be combined with indirect constraint handling techniques that focus on the remaining constraints. When knowledge about the constraints is incorporated using the right techniques, EAs can be effective constraint solvers [58].

This chapter provided a brief overview of the pros and cons of local search and evolutionary algorithms, and about the possible benefits of mixing these heuristic methods. Furthermore, an explanation of the different stages of an EA was given, detailing the decisions that need to be made when developing an EA. Also, a specific EA that was developed in recent years was discussed, called GOMEA. Finally, some of the additional issues that arise with constraint satisfaction problems (which TUSS is) were addressed. Altogether this chapter introduced the necessary background information regarding EAs and local search for this research. The next chapter discusses TUSS in more detail and lays out the local search method that the NS is currently using.

Chapter 3

Train Unit Shunting and Servicing Problem

Before the evolutionary algorithm developed in this research can be described, a more detailed explanation of the Train Unit Shunting and Servicing (TUSS) problem is required. This chapter provides such an explanation. Furthermore, components of Hybrid Integrated Planner (HIP), the local search algorithm implemented by the NS, are an integral part of the evolutionary algorithm. Therefore the mechanisms of HIP are also discussed in this chapter. Both the problem description and the breakdown of HIP are largely based on the thesis of van den Broek [14], the creator of HIP.

3-1 Problem description

An elaboration of the problem is needed to understand how the heuristic algorithms for TUSS work. This elaboration is split into parts. First, preliminaries concerning the location and trains are discussed. Next, a brief description of the four subproblems is given: matching, servicing, parking, and routing.

3-1-1 Preliminaries

A shunting yard, or service site, consists of a set $\mathcal{T}$ of *tracks* that are connected by *switches*. A track $\tau \in \mathcal{T}$ has a *length* of l_τ and two endpoints referred to as the $A-$ and $B-$*side* of τ. Tracks approachable from both sides are called *free* tracks, while *LIFO* tracks are only accessible from one side, with the other side being blocked by a bumper. A shunting yard is connected to the railway network through one or more tracks known as *gateway*-tracks.

Each service task $\sigma \in \mathcal{S}$ that needs to be performed has a duration d_σ and a set of tracks $\mathcal{T}_\sigma$ on which the tracks can take place. Service tasks require a certain *facility* that is located along a track, such as a cleaning platform. Only a single train can be processed at a time in each facility.

The trains that enter and exit the service site consist of one or more *train units*, which are bi-directional and self-propelling vehicles that move without a dedicated locomotive. The set TU comprises all train units at the service site. Train units $tu \in TU$ are classified according to *types* (t_{tu}) and *subtypes* (st_{tu}).

Incoming trains can be split and combined to form new train compositions. These compositions are referred to as shunt trains, and every train unit is always part of a shunt train during its stay at a service site. Train units can only be combined if their train types are equal.

A *saw move* defines a reversal of the direction of a train. For instance, if a train parks at a LIFO-track, it needs to reverse its direction to be able to leave the track again. This manoeuvre is time-consuming and hence avoided by human planners if possible.

Finally, a *crossing* is defined as the scenario in which the path of a moving train is blocked by one or more other trains. This is undesirable and a plan that contains one or more crossings is infeasible.

3-1-2 Matching

With these preliminaries out of the way, the four subproblems of TUSS can be explained. The first subproblem considered is the matching problem: given are a set of arriving trains and a set of departing trains. Each arriving train has an arrival time and consists of one or more train units. A departing train departs at a specified time and specifies a sequence of one or more train subtypes. The arrivals and departures are mixed over time, i.e. a departure can happen before all trains have arrived.

The objective is to match each train unit in an arrival time to a specific position in one of the departing trains such that the train subtypes match and the train unit enters the service site before departure. When two train units are coupled at arrival and are assigned to different departing trains, they have to be split at the service site. Finally, the service site is assumed to be empty at the start and end of the day (i.e. all trains that arrive also depart again that day).

3-1-3 Servicing

In the servicing or task scheduling problem each train unit has a (possibly empty) set of service tasks, with a respective duration, that need to be performed during its stay at the shunting yard. Each service task requires a certain resource for its entire duration; the availability of resources is limited. No precedence relation exists between service tasks, i.e. it does not matter in what order service tasks are performed. The objective of the task scheduling problem is to construct a schedule such that all tasks are completed and all trains can depart on time.

3-1-4 Parking

A train that moves to a track τ is added in front of one of the two sides of the sequence of trains already parked on the track. The side is fixed if τ is a LIFO-track, else it depends on the

arrival side of the train. The main objective of the parking problem is to place all shunt trains on tracks such that at any moment in time the combined train length does not exceed the length of the track, while simultaneously ensuring that each shunt train has an unobstructed path off the track when it has to depart. An additional constraint is that parking, splitting and combining is not allowed at the gateway tracks, which in particular means that all shunt trains of a departure composition should be parked on one track, in the correct order, to combine the train units.

3-1-5 Routing

Finally, the aim of the routing problem is to find for each train movement an unobstructed path that minimises the movement duration. For service sites operated by the NS, the duration is estimated based on the number of tracks, switches and required saw moves on the path.

Simultaneous movements are not allowed at the service site due to safety regulations. This simplifies the routing problem since routes cannot intersect, and thus the computation of one route is independent of other routes. Furthermore, it implies a linear ordering can be defined on all movements at a service site.

3-2 Hybrid Integrated Planner (HIP)

Now that TUSS has been detailed the mechanisms of HIP are discussed. The current version of HIP closely resembles it as it is described in [14]; however, it now uses Variable Neighbourhood Search (VNS) instead of Simulated Annealing (SA) as a search mechanism. This section gives a brief overview of HIP: for the interested reader, the thesis of van den Broek [14] contains a more elaborate explanation of HIP.

3-2-1 Objective function

The goal of HIP is to find a feasible solution to TUSS, making it a Constraint Satisfaction Problem (CSP). To solve CPSs with heuristic search methods a mix of direct and indirect constraint handling techniques can be employed, as is elaborated on in Section 2-3. This is exactly what HIP does: a subset of constraints are ensured to always hold, while the remainder of constraints are incorporated as penalties into an objective function.

The following constraints always hold in any plan created by HIP:

- Each service resource has at most a single task at any time.
- No simultaneous movements are ever allowed.
- A proper matching of arriving and departing trains is ensured during the algorithm, i.e. no type mismatch can occur.

The other constraints in TUSS are incorporated in the objective function as penalties, and include:

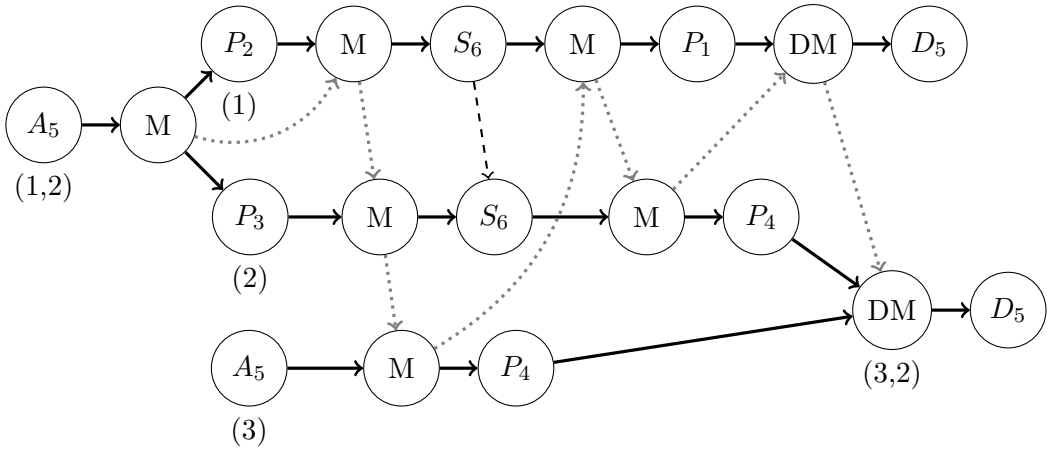

Figure 3-1: An example STAG. The subscripts at the arrival, parking, and servicing nodes indicate the track that activity took place on.

- The number of crossings.
- The number of occasions in the shunt plan where the combined train length of shunt trains occupying track τ exceeds the length l_τ.
- The delay of departure trains.
- The delay of arrival trains. This happens when an arrival train arrives during the movement of another train since there can only be one movement at a time. This means the arrival train has to wait at the gateway track until the other movement is completed.

A shunt plan is feasible if all of the above values are zero. During the search a weight is given to each of these values to provide a measure of the quality of a solution. The resulting number is the objective value and is used to guide the search: a lower value is supposedly 'closer' to a feasible solution.

3-2-2 Shunt Train Activity Graph

Like an evolutionary algorithm, a local search algorithm needs a solution representation that properly captures all important aspects of the solution, while simultaneously allowing for easy modification through the local search operators. To this end, HIP uses a Shunt Train Activity Graph (STAG) as a solution representation.

Figure 3-1 shows an example STAG. A STAG can have six different types of nodes:

- Arrival (A) node: represents the arrival of a train at the shunting yard.
- Movement (M) node: represents a movement of a train at the shunting yard.
- Parking (P) node: represents a parking action of a train at the shunting yard.
- Service (S) node: represents a train being serviced at the shunting yard.

- Departure movement (DM) node: represents a special type of movement from a train at the shunting yard to its designated departure track. This node always precedes a departure node.
- Departure (D) node: represents a train departing from the shunting yard. Each departure node corresponds to a specific outgoing train.

The arcs in the graph indicate the precedence relations between the nodes. The solid, black arcs represent the order of actions for an individual train unit. A path over these arcs from the arrival node to the departure node shows the actions for that train unit in the shunting plan.

The numbers between parentheses underneath some of the nodes refer to the train unit(s) that that node (and succeeding nodes) applies to. When a movement has more than one outgoing solid, black arc, it represents a splitting of that train into multiple units. In the example figure, a train arrives consisting of train units 1 and 2, and these are then split and parked on separate tracks (i.e. unit 1 is parked at track 3, unit 2 is parked at track 2). Similarly, a (departure) movement node with multiple incoming solid, black arcs represents a combining of multiple train units into a single train (in the example figure, train units 2 and 3 are merged into one train before departure).

The dashed, grey arcs between movement nodes indicate the ordering of movements in the shunting plan. This is a strict ordering since at any time only one movement is allowed to take place on the shunting yard.

Finally, the black dotted arcs represent the order in which service tasks that are assigned to the same resource are performed.

From the four subproblems that TUSS is made up of (matching, servicing, parking, and routing), the STAG captures the first three. It captures matching because each departure node corresponds to a particular outgoing train, so the train unit(s) that have a path to that departure node are matched to that departure. It captures servicing because it shows for each train unit that needs servicing at which track it is being serviced, and in what order the service tasks are performed. It captures parking because it shows for each train unit on which track(s) it is being parked. However, the STAG does not capture routing or times, so these need to be calculated separately.

3-2-3 Routing and time

Once a STAG is created for a problem, the routes for each movement can be calculated. A precedence relation is defined between the movement nodes, and there are no simultaneous movements, so the routes for the movements can be calculated in order. Because of the precedence relations in the graph, the state of the service site at the time of any movement is known (i.e. which trains are parked where), and thus routes can be accurately calculated. Previously calculated routes are stored in cache and used to guide the search for new routes using the informed search algorithm A* [61].

Once the routes and their respective durations have been computed, the start- and end times of the nodes can be calculated. The earliest start time of a node is determined as the maximum end time over all predecessor nodes in the activity graph. Once times have been added to the nodes, the result is a valid plan that can be used in practice if it is feasible.

Subproblem	Operator	Details
Routing	Movement shift	Change the order in which movements take place, with the constraint that the resulting plan is still valid
	Movement merge	Sometimes a plan contains unnecessary movements that can be merged
Parking	Parking reassignment	Change the track and arrival side of a parking activity
	Parking swap	Swap the track and arrival side of two parking activities that overlap in time
	Parking reposition	Split a parking node into two, with a movement node in between, and one of the parking nodes is assigned to a different track than the current parking node
Servicing	Resource order swap	Swap the order of the consecutive tasks assigned to the same resource
	Train order swap	Swap the order of two consecutive service tasks of the same shunt train
	Resource reassignment	A service task is assigned to a different resource
Matching	Matching swap	Swap the assignment to positions in departing trains of two (partial) shunt trains

Table 3-1: Local search operators used to adapt existing plans.

3-2-4 Initial solution

The previous sections discuss the representation of a plan for TUSS in HIP. The algorithm also needs a starting point, which is why a method for creating initial solutions is needed. HIP uses heuristics to create an initial solution. The following steps are taken, in this order, to construct an initial solution:

1. A valid matching is created using the polynomial-time Hopcroft-Karp algorithm [62].
2. Service tasks are scheduled in order of increasing due date to an arbitrary resource. The due date is the departure time of the corresponding train, which is already known because a matching has been made.
3. Movements from and to each service task are assumed in the initial heuristic, that is, each shunt train will be parked between each service task.
4. For each parking interval, a random track is picked from the tracks with sufficient free space during that interval, if such a track exists. Else, any track is picked.

3-2-5 Search

Once an initial solution has been created, the local search can be started. HIP uses Variable Neighbourhood Search (VNS) which has two phases. In phase one the search only accepts changes that improve a solution, steering it towards a local optimum. In phase two the search

makes random changes to a solution that do not necessarily improve it, to get out of a valley in order to find the global optimum (i.e. a feasible solution in the case of TUSS). The local search algorithm alternates between these two phases.

The defining aspect of HIP is the local search operators that are used to change an existing plan, which are related to the search neighbourhoods that are defined. It is beyond the scope of this thesis to discuss these neighbourhoods in detail, but a short overview is given in Table 3-1.

These search neighbourhoods are used to change existing solutions in an attempt to find a feasible solution. Given a solution, HIP checks which operators can be applied without violating any of the constraints that should always hold, described in Section 3-2-1. From the possible operators that can be applied, a random one is chosen. If the algorithm is in phase one and the change deteriorates the solution, it is reverted. In phase two it is always retained. In case the change is reverted, a different operator is chosen to evaluate if this does yield an improvement. The algorithm continues like this until a stop criterion is met, which can be a time or evaluation limit, or a feasible solution.

This chapter provided an explanation of TUSS, detailing its core concepts. The core concepts of HIP, the local search algorithm created by the NS, are also detailed in this chapter. The next chapter explains the evolutionary algorithm created in this research.

Chapter 4

An evolutionary approach to solve TUSS

The previous chapters discussed possible benefits of Evolutionary Algorithms (EAs) over local search, how EAs and local search can be combined to form a hybrid method, and the general structure of an EA. Furthermore, the Train Unit Shunting and Servicing (TUSS) problem and Hybrid Integrated Planner (HIP), the local search method currently being developed by the NS, were discussed. With this, all necessary background information has been presented such that the focus can now be shifted to Conflict-Based Crossover (CBC), the EA developed in this research.

One of the most important advantages of an EA over local search is that it is (supposedly) better at exploring the search space, contrary to local search which has a higher probability of getting stuck in a local optimum. CBC is developed with this notion in mind, to attempt to explore the search space in an effective manner that would be hard to mimic for local search methods.

The most important contributions of CBC are the following. A diversity metric is created which quantifies the extent to which solutions differ. This can be useful for analysis of the progression of the algorithm and it is used for the parent-selection phase, which is another contribution of CBC: diversity-based parent selection. The most important contribution, however, is the creation of a crossover operator and an accompanying heuristic that guides the crossover; the name of the algorithm is derived from this heuristic. Crossover combines two solutions to form a new valid solution; solutions to TUSS are complex plans, and as such combining two of these plans to form a new plan is not a trivial task. The crossover operator accomplishes this task. Furthermore, the goal of crossover is to create a new solution that is located in a promising area of the search space. The heuristic that guides the crossover is designed to achieve this.

Figure 4-1 shows an overview of the different components of CBC in a flow diagram. The remainder of this chapter discusses the different components in more detail. Section 4-1 elaborates on components that are not innovative but are a vital part of the algorithm.

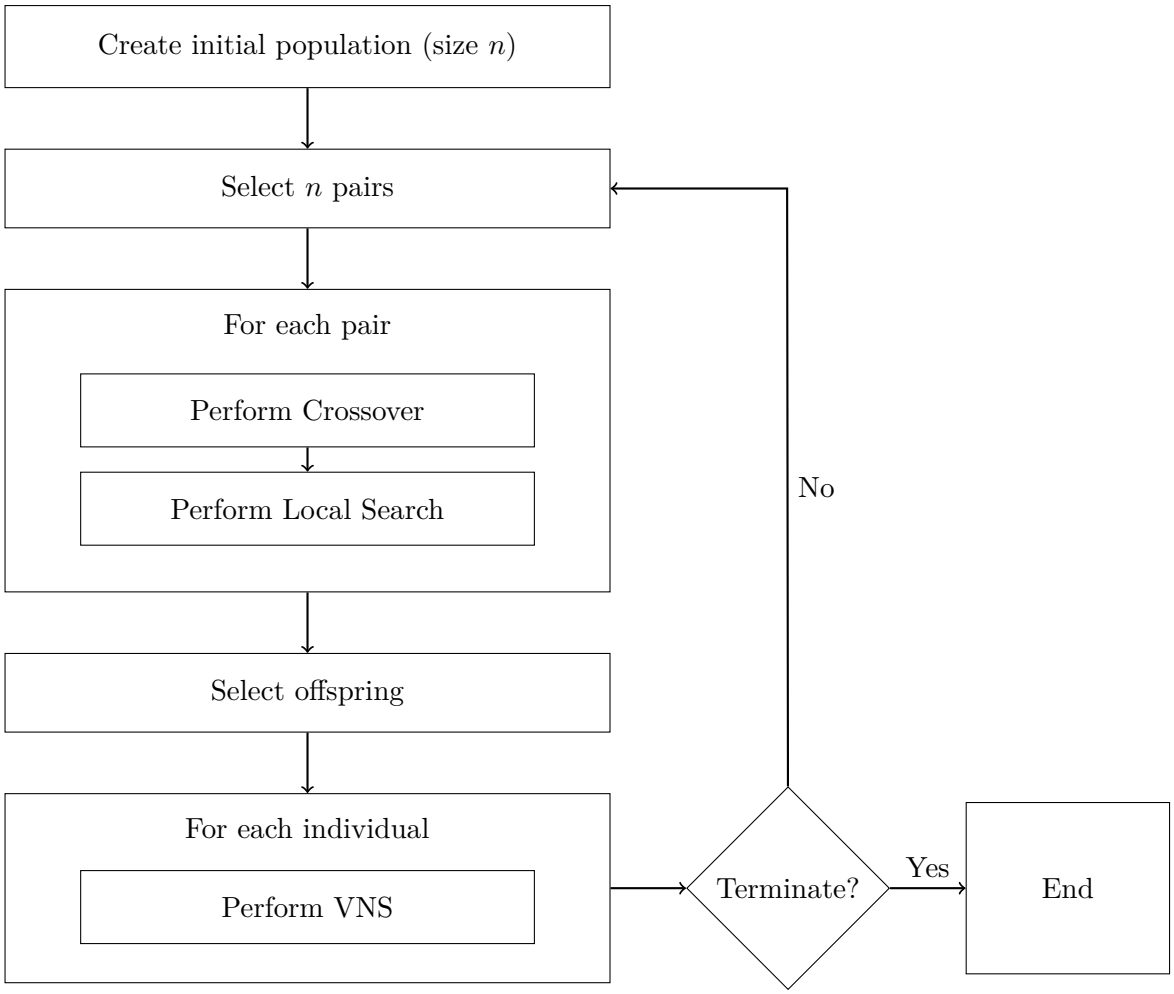

Figure 4-1: Overview of CBC

These include the representation and evaluation of TUSS (used among other things to decide whether to terminate), the initialisation of the population, the way in which offspring is selected, and how and why CBC is hybridised. Section 4-2 explains the diversity metric that is created in this research, which is used for selecting parents for crossover, discussed in Section 4-3. Finally, Section 4-4 discusses the most important contribution of this research, which is how crossover is applied.

4-1 Literature-based components

This section discusses components of CBC that are based on literature, and are therefore, contrary to the components discussed in the remaining sections of this chapter, not part of the innovative contributions of this research. They are mentioned here to provide a complete description of the algorithm.

4-1-1 Representing and evaluating TUSS

The first step in the development of an EA is defining a solution representation and a measure for the quality of solutions. Since this work builds upon HIP it uses the same representation that HIP does, which is the Shunt Train Activity Graph (STAG) presented in Section 3-2-2. This is both practical (i.e. the implementation of the EA can use the codebase of HIP as a starting point, instead of having to implement the algorithm from zero) and efficient (i.e. STAGs have been extensively tested and tuned in HIP and appear to work well for TUSS). Most importantly, two different STAGs can be combined to form a new STAG, which is the goal of the crossover operator in an EA. Section 4-4 details how this is done in CBC.

A similar argument applies to the evaluation function (also called fitness function or objective function). HIP uses an evaluation function, presented in Section 3-2-1, to qualify solutions and to guide the search. A lot of testing has been done to develop and tune this evaluation function, to optimise the performance of HIP. Since CBC is a hybrid method that also employs local search, the evaluation function is expected to also work well for CBC. Therefore, CBC uses the same evaluation function to measure the quality of solutions.

4-1-2 Initial solutions

To be able to run any EA it has to acquire an initial population. To do so, initial solutions to the problem at hand need to be created. In the case of TUSS, heuristics can be used to construct an initial, valid solution. HIP contains heuristics to do so (see Section 3-2-4), and CBC uses the same heuristics to create initial solutions. Each solution is created with a different random seed, thereby ensuring that different initial solutions are created.

4-1-3 Selecting survivors

CBC creates n new solutions in the crossover stage (given a population size of n). This means that there are $2n$ plans after this stage. The population size in CBC is kept constant throughout generations. To achieve this, only half of the $2n$ solutions can survive into the next generation. This section discusses how this selection is made in CBC.

CBC uses a well-known method called tournament selection. All $2n$ individuals are randomly divided into groups of size t, where t should be a power of 2. In each group, the 'best' individual (i.e. the one with the lowest fitness value) is selected to survive into the next generation.

This process of splitting the $2n$ individuals into groups and selecting the winner of each group for survival needs to be done $0.5t$ times, such that n individuals are selected for survival.[1] Group sizes of 2 and 4 are common practice. When a group size of 4 is used, the whole process is executed twice, which implies that the best solution found so far will be in the next generation twice (because the best solution will always be the winner of its group). This lowers diversity and speeds up convergence. Because relatively small population sizes are used in CBC, a group size of 2 is opted for.

[1] With a group size of t, there are $\frac{2n}{t}$ groups, and just as many survivors are selected. Simple math shows that the process then needs to be executed $0.5t$ times to yield n survivors: $\frac{2n}{t} \cdot \frac{t}{2} = n$

4-1-4 Hybridisation

EAs can be hybridised by adding local search to the algorithm, which is what is done here. CBC is hybridised to ensure that the crossovers are performed on local optima, thereby expecting to have more effective crossovers. There are two stages in the algorithm where a local search meta-heuristic is applied, which are discussed below.

Hill-climbing

A common way to hybridise EAs is to add a local search heuristic after the variation phase. This is the first stage where CBC is hybridised in this research. After crossover is performed, a local search heuristic is applied to each newly created solution.

This local search heuristic is identical to HIP in everything but the search strategy employed. HIP uses Variable Neighbourhood Search (VNS) as a search strategy, meaning that it can also accept changes that worsen the current solution (explained in more detail in Section 3-2-5). The search strategy used here is a hill-climbing strategy, meaning that only changes that increase the current solution are accepted. This way the solution that is created after crossover is adapted to move to a local optimum.

A hill-climbing strategy is used because it gives a clear endpoint to the local search: it stops when no further improvement can be made. This implies that the length of the search after each crossover varies because some newly created solutions will have a lot of room for improvement while others will not. If a search strategy like VNS is used here, an arbitrary endpoint for the search would have to be picked and this would be the same for every solution, which would cause some searches to stop prematurely, while going too long in other cases.

As will be apparent during the discussion of the results of CBC, this local search step is of vital importance to the success of the algorithm. The crossover that is performed can introduce new conflicts that were not present in either of the parent solutions. However, sometimes these conflicts can be resolved easily with small changes, which is an area where local search heuristics excel.

Searching the neighbourhood

The last step of a generation in CBC is applying HIP search to each selected survivor for a predetermined amount of function evaluations. The motivation for adding VNS search to the algorithm is that crossover makes large changes to existing solutions, thereby taking large leaps in the search space. This means that when it is close to a global optimum (i.e. a feasible solution) it is likely to 'jump' over that optimum. Local search makes small, incremental changes and is therefore less likely to 'miss' a global optimum when it is close.

The reason for using VNS here instead of hill-climbing search (used after crossover) is that every solution already had hill-climbing search applied to it until no further improvement exists, so every solution is guaranteed to be at a local optimum. Thus hill-climbing would not yield any improvement at this point. However, one such local optimum might be close to a global optimum and, contrary to hill-climbing, VNS is potentially able to reach it by accepting a change that worsens the current solution and thereby going through the valley that lies between the current solution and a feasible one.

Finally, this VNS step is the reason that mutation is not included in the EA. Mutation makes a random change to a solution with a certain probability, which is very similar to what VNS does by accepting a change that worsens the solution. Therefore mutation does not add any value to this algorithm.

4-2 Diversity

Diversity is a way to measure the spread of the individuals in the population. The use cases for a proper diversity measure in CBC are twofold. First, it can be used to analyse the performance of various methods, as it gives some insight into the rate of convergence of an EA. Second, the parent selection described in Section 4-3 uses the diversity measure presented here to select parents.

String representation

To be able to measure the diversity in a population, a method is needed to quantify how different two solutions are. To do so with TUSS, CBC transforms the STAG representation for solutions to TUSS presented in Section 3-2-2 into a string, effectively giving a string representation of a solution to TUSS.

The string representation works as follows: for each train unit in a solution, each activity that is not a movement activity is represented as a letter (signifying the activity) and a number (signifying the track the activity took place on). There are four types of activities (movement activities are not included because they take place between every activity):

- Arrival activity: represented by the letter 'A'.
- Parking activity: represented by the letter 'P'.
- Servicing activity: represented by the letter 'S'.
- Departure activity: represented by the letter 'D'.

For instance, the string representation for the STAG in Figure 3-1 is as follows (corresponding to units 1 through 3, in that order):

$$A5P2S6P1D5$$
$$A5P3S6P4D5$$
$$A5P4D5$$

It should be noted that this abstracts over even more information than the STAG (since it does not include movement activities), and it is therefore possible that two different solutions have the same string representation. However, it still provides a good basis to quantify the difference between two solutions.

Levenshtein distance

To quantify this difference the Levenshtein distance (or edit distance, i.e. the minimum number of insertions, deletions and substitutions required to convert one string into the other) is calculated between the string representations of two solutions (each string representation has its train units in the same order, so the Levenshtein distance is effectively calculated per train unit and then summed). The resulting number indicates how different two solutions are.

To calculate the diversity of the entire population the diversity between all pairs of solutions is calculated and summed. When comparing the diversity between populations of different sizes, this number can be normalised by dividing it by the total number of pairs in a solution, yielding the average diversity for one pair of solutions.

4-3 Matchmaking for parents

The parent selection phase is used to select individuals that are paired together to create offspring. In CBC, given a population of size n, n exclusive pairs (i.e. no pair can be selected twice, however, one individual can be in multiple pairs) of parents are selected for the variation stage. Two different methods to select parents are implemented and tested.

The first method is simple and serves as a baseline method: pairs of parents are selected randomly. The only constraint imposed on the random selection is the aforementioned one; each pair can only be selected once.

The second method aims to keep the diversity in the population high to prevent premature convergence. It does this by selecting the n pairs of solutions in the current population that have the highest diversity between them. The intuition behind this choice can best be reasoned about from an opposing angle. When two solutions that are the same (or very similar) are combined to form a new solution, the new solution will be the same (or very similar) to the original solutions, thus decreasing the diversity. On the other hand, solutions that have a high diversity between them are very different and reside in different parts of the search space. Since parts of both solutions are used to create a new solution, the expectation is that this new solution will reside in yet another (unexplored) part of the search space, thereby increasing the diversity and exploiting the exploratory nature of EAs.

4-4 Creating offspring

Once the pairs of parents have been selected these are recombined to form new solutions, which is the core of any EA. In CBC, crossover is used to mix two solutions to form a new solution, and this section explains how crossover can be applied in the case of solutions for TUSS that are represented by STAGs. This entails both the method for combining two solutions to form a new valid solution and the method for deciding which parts of each parent solution should be used in this process. Both methods are presented here.

It should be noted that the crossover operator currently does not support splits and combines. The STAG in Section 3-2-2 (Figure 3-1) does contain splits and combines, which can be recognised by movement nodes that have multiple outgoing and incoming arcs, respectively.

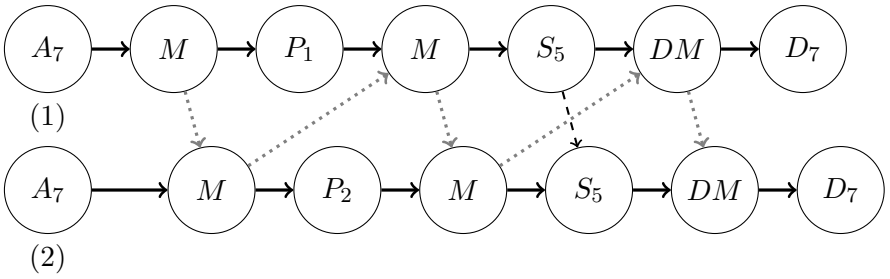

Figure 4-2: Solution A

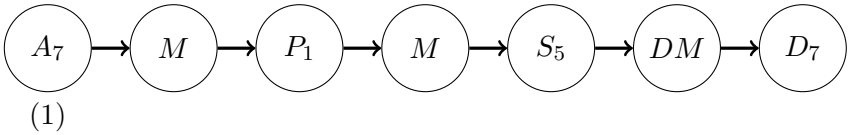

Figure 4-3: Schedule for unit 1 in Solution A, which is extracted by taking the path over the solid, black arcs from the arrival node of unit 1 in Figure 4-2 to its departure node.

Since splits and combines are currently not supported by the EA, that effectively means that each train is assumed to consist of a single train unit, and all movement nodes in the STAGs in this chapter have one incoming and one outgoing arc. Section 6-1-1 discusses how splits and combines can be integrated with CBC in the future.

4-4-1 Recombining two solutions

To perform crossover in TUSS, the schedule of each individual train unit in a STAG is viewed as a variable. The schedule of a train unit corresponds to a path in the STAG over the solid, black arcs from its arrival node to its departure node. Figure 4-2 shows an example STAG and Figure 4-3 shows the corresponding schedule of unit 1.

Crossover is performed by selecting variables (i.e. schedules for train units) from both solutions (such that every variable is selected exactly once) and using these to form a new STAG. Figures 4-2 and 4-4 show examples of two STAGs that can be combined. Suppose variable 1 (i.e. train unit 1) is selected from Solution A and variable 2 (i.e. train unit 2) is selected from Solution B. The new solution is then represented by the STAG in Figure 4-5.

The previous paragraphs discuss the gist of the crossover method used in CBC. However,

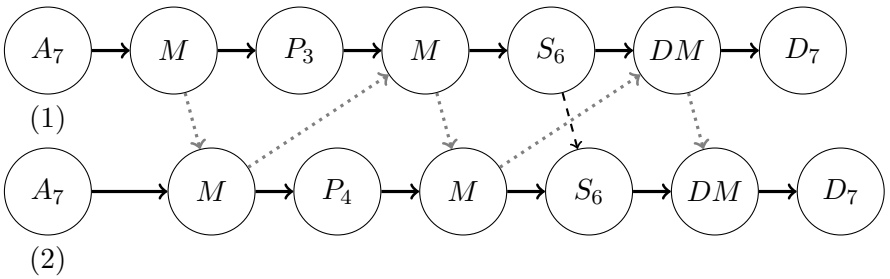

Figure 4-4: Solution B

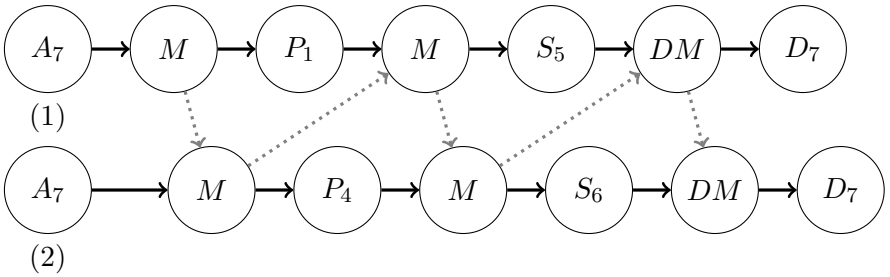

Figure 4-5: This solution is obtained by taking train unit 1 from the solution in Figure 4-2 and train unit 2 from the solution in Figure 4-4 and combining them to form a new solution. Unit 1 now parks at track 1 (as it does in Solution A) and unit 2 now parks at track 4 (as it does in Solution B). Furthermore, the units now have their service tasks performed at different service stations.

the attentive reader may have already noticed that the order of movements and service tasks is not apparent from the recombination strategy described above. To order the movements and service tasks in the new solution, their times in their original solution are used. In the example that is used here, the service tasks in the new solution are at different service stations, which means they do not require ordering (since the ordering of service tasks only applies to service tasks that are performed at the same resource). For the movements, it means that the times of the movement actions of unit 1 are taken from Solution A and the times of the movement actions of unit 2 are taken from Solution B. These times are then used to order the movements in the new solution. Here it is assumed that their order stays the same (this cannot be deduced from the STAGs depicted here because times are not shown).

Note that the ordering of these movements can be problematic when two solutions are combined that have a different matching. The level of detail that is involved with explaining this issue (and its solution) is not in line with the rest of this thesis and is not relevant for the remaining content in this chapter, and is therefore not discussed here. However, the interested reader can refer to Appendix B for an explanation of this problem.

Another aspect that has not been addressed yet is the matching. The matching of incoming trains to outgoing trains should still be valid after crossover, and two solutions that are mixed can have a different matching. The solution is simple: in the crossover phase, the matching is taken from one of the two parent solutions (since both have a valid matching, this ensures a valid matching in the new solution). This means that the departure and departure movement nodes in the STAG are all taken from one solution during crossover. Since the arrival nodes are the same for all solutions (arrival times and tracks are part of the input), this effectively means that during crossover only the parts between the arrival nodes and the departure nodes of the STAGs are changed.

Once a valid STAG is constructed, the exact times for each node can be calculated. Furthermore, the routes for the movements can also be calculated at this point. This happens in the same fashion as it does in HIP, which is explained in Section 3-2-3.

4-4-2 Shunt unit selection

The previous section showed how two solutions for TUSS that are represented by STAGs can be combined such that a new, valid solution originates. However, it did not discuss how to select which variables to take from which solution. This is arguably one of the most important design choices of the algorithm. If two solutions can be combined in such a way that the good parts of both solutions are kept and the bad parts are discarded, this is expected to yield new solutions that can be promising.

Two methods for selecting variables are presented here. One acts as a baseline method while the other method is aimed at maximising the effectiveness of the crossover. It aims to do so using heuristics that are devised based on domain knowledge.

Uniform

The baseline method that is used for selecting variables for crossover is uniform crossover. This is based on uniform crossover in genetic algorithms, where each gene is selected at random from either of the parents. It behaves in the same fashion here: for each variable (i.e. train unit) it is selected at random from either of the parents, with equal probability. So no domain knowledge is used and the expectation is that a method that effectively selects variables based on smart heuristics should be able to outperform uniform crossover.

Conflict-Based Crossover

Conflict-based crossover is the heuristic devised for effectively selecting variables and also names the EA in this research. To explain this heuristic, a few concepts need to be explained first, after which the mechanisms of conflict-based crossover are discussed. The section ends with an example detailing the steps that are taken during this crossover.

Conflicts. Section 3-2-1 describes the objective function that is used in HIP and CBC to qualify solutions. Any event in a plan that increases the objective value, is called a conflict. Examples of this are a departure delay or a crossing. For each train unit, the number of conflicts it is involved in can be calculated. In other words, this measures how much each train unit contributes to the objective value. This measure is used in one of the criteria for variable selection.

Neighbour count. When two trains are parked at the same track at the same time these are considered neighbours. The neighbour count of train unit u with a subset S of other train units is defined as the total amount of neighbours that u has in S in the current plan. This measure is also used in one of the criteria for variable selection.

Overlap. Finally, when two trains both park at track x at some point in time (they do not have to be parked there at the same time, making this distinct from being neighbours), they are said to have overlap. The total overlap between a train unit u and a subset S of

train units is then defined as the sum of the overlap between u and all units in S. Again, this measure is used in variable selection.

Now that the necessary concepts have been explained, the variable selection criteria used in conflict-based crossover can be laid out. The following criteria are used, in order:

1. First, all train units that are involved in fewer conflicts in one of the parent solutions are selected from that solution. This is the most important step, and its intuition is simple: units that are involved in fewer conflicts are likely to have a better schedule. By making this selection, the idea is to take parts of both solutions where they are evidently superior to the other. After this step, the only train units that remain for selection are the ones that are involved in the same amount of conflicts in both parents.

2. For these remaining units, for each parent, the neighbour count of a unit with the units already selected from that parent is calculated. Only selected units are considered because those are the only ones that are certain to be in the new solution. Units that have not been selected from a parent yet might never be, and then it is irrelevant whether they neighbour selected units or not since this does not influence the new solution. For each unit the neighbour count is used as follows:

    - If the conflict cost for a unit is zero, it is selected from the parent where it has the highest neighbour count.
    - Else, it is selected from the parent where it has the lowest neighbour count.
    - The reasoning behind this is that if a unit is involved in zero conflicts currently, it likely had a good synergy with the units around it. By maximising the neighbour count, you keep the units around it the same as much as possible. Similarly, if the conflict cost is greater than zero, you want to keep as few neighbours the same as possible because these neighbours might have caused conflicts.

3. If the neighbour count is also equal, the unit is selected from the solution where it has the least overlap with all the units that were already selected (the same argument as in the previous point applies here as to why only selected units are considered). The intuition behind this is that if there is little overlap, the chances of the newly added unit creating conflicts is also smaller.

4. Finally, if the overlap is also equal, the unit is selected from either parent at random.

Because the explanation above may be somewhat abstract, an example is now presented which steps through the different parts of the variable selection process. It is an example with four train units that takes place on the example location depicted in Figure 4-6. Table 4-1 shows the order in which the trains arrive and depart. This implies the matching is predetermined, which is done for the sake of simplicity in this example.

In the example, all arrivals take place before the first departure, meaning that all train units will be parked at the shunting yard at one point in time. Suppose there are two parent solutions that need to be combined to form a new solution. Figures 4-7 and 4-8 show the configuration of the trains on the track after the last arrival and before the first departure for Parent A and Parent B, respectively. The STAGs corresponding to the respective solutions are omitted here for brevity.

Order of arrival	Order of departure
1	3
2	4
3	1
4	2

Table 4-1: The order of arrival and departure of the train units in the example (i.e. train unit 1 arrives first while train unit 3 departs first).

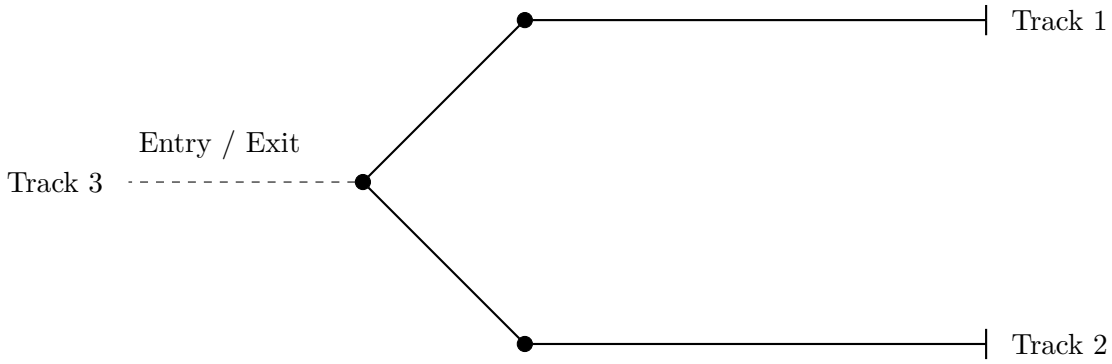

Figure 4-6: Example location. Trains enter and exit the location through track 3, and are only allowed to park on Track 1 and Track 2. A train cannot move from Track 1 to Track 2 or vice versa.

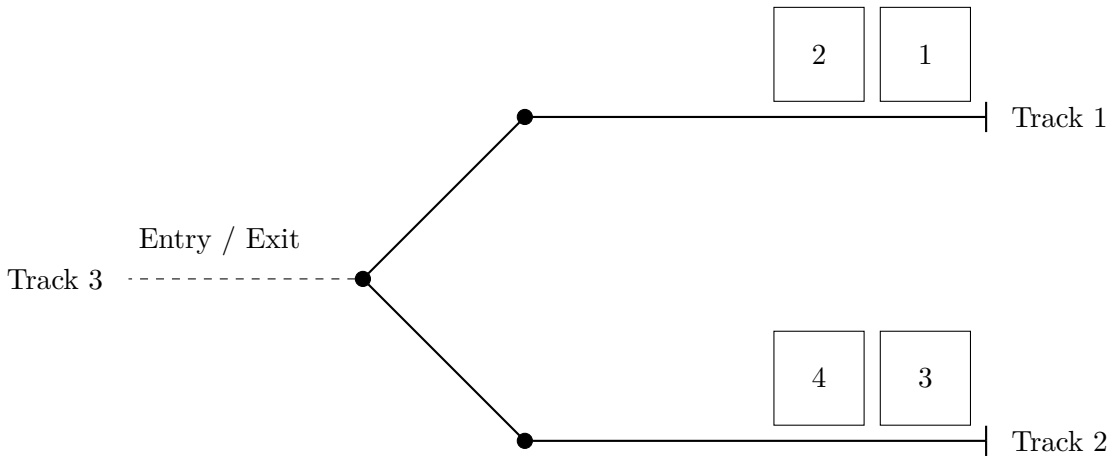

Figure 4-7: Depiction of the tracks that the units in Parent A are parked on when they are all present at the track. Conflicts can easily be shown from this image. For instance, unit 3 departs before unit 4, yet unit 4 is blocking unit 3 when it wants to move to Track 3 for departure. This yields a conflict.

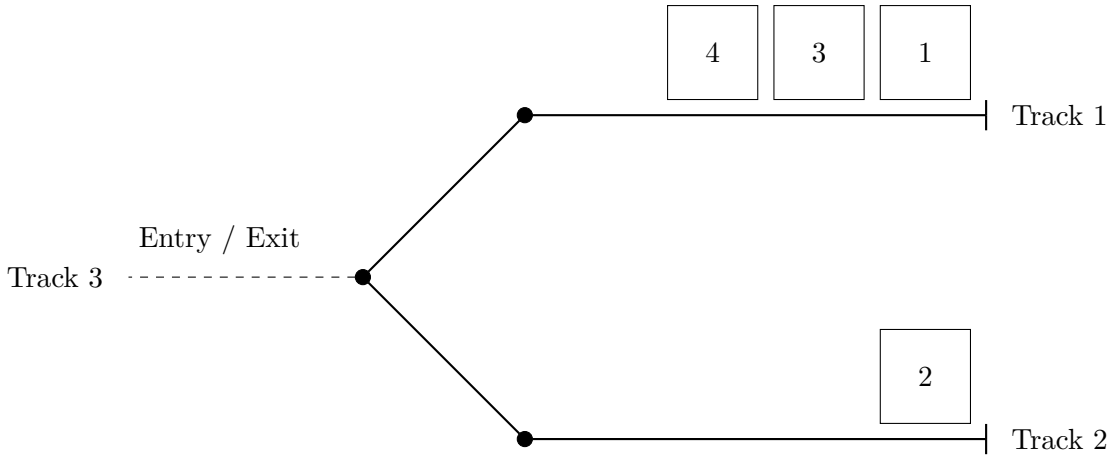

Figure 4-8: Depiction of the track that the units in Parent B are parked on when they are all present at the track.

Using these configurations, we can now show how the conflict-based crossover heuristic makes its decisions to select train units from one of the parents. The first step is to calculate the conflicts each is involved in. Table 4-2 shows the results of this calculation.

Train Unit	**Parent A**		**Parent B**	
	Conflicts	Specification	Conflicts	Specification
1	1	Crossing with 2	0	N\A
2	1	Crossing with 1	0	N\A
3	1	Crossing with 4	1	Crossing with 4
4	1	Crossing with 3	1	Crossing with 3

Table 4-2: Conflicts for each train unit in Parent A and Parent B in the simple example. For example, in Parent A train unit 1 is blocked by train unit 2 when it wants to depart (since unit 1 departs before unit 2). This is a crossing conflict. It is left as an exercise for the reader to verify the other conflicts in this table.

From this table, we can conclude that units 1 and 2 are selected from Parent B since they are involved in fewer conflicts in Parent B than in Parent A. For units 3 and 4, the number of conflicts they are involved in is equal in both parents, so these are not selected in this stage.

Instead, the neighbour count is calculated for these units. For unit 3, the neighbour count for Parent A is 0, since no units have been selected from Parent A yet. From Parent B, units 1 and 2 have been selected. Unit 3 is parked on the same track as unit 1 in Parent B (see Figure 4-8) and as such the neighbour count for unit 3 in Parent B is 1. This is larger than the neighbour count for unit 3 in Parent A, and because unit 3 is involved in more than zero conflicts, it is selected from the solution where it has the lowest neighbour count, which is Parent A.

Unit 4 is parked alongside unit 3 (which was just selected from Parent A) in Parent A, so the neighbour count with Parent A is 1. In Parent B, unit 4 is parked alongside unit 1, so the neighbour count is also 1. Since the neighbour count is equal, this does not yield a decisive answer, and we calculate the overlap.

In Parent A unit 4 is parked on track 2. From Parent A unit 3 is selected, which is also parked on track 2. From Parent B units 1 and 2 are selected. Unit 2 is parked at track 2 as well. That brings the overlap to a total of 2 for unit 4 from Parent A.

In Parent B unit 4 is parked on track 1. So there is no overlap with unit 3 (which was selected from A and parked on track 2). From the selected units from Parent B, unit 1 is also parked at track 1, while unit 2 is not. This brings the overlap to a total of 1 for unit 4 from Parent B.

Because the overlap in unit 4 from Parent B is smaller, it is selected from this solution. This concludes the variable selection process for this example. Units 1, 2, and 4 are selected from Parent B, whereas unit 3 is selected from Parent A. Mixing both parents in this fashion yields the configuration of trains on tracks depicted in Figure 4-9. All conflicts have been solved by the crossover, and a conflict-free solution emerged.

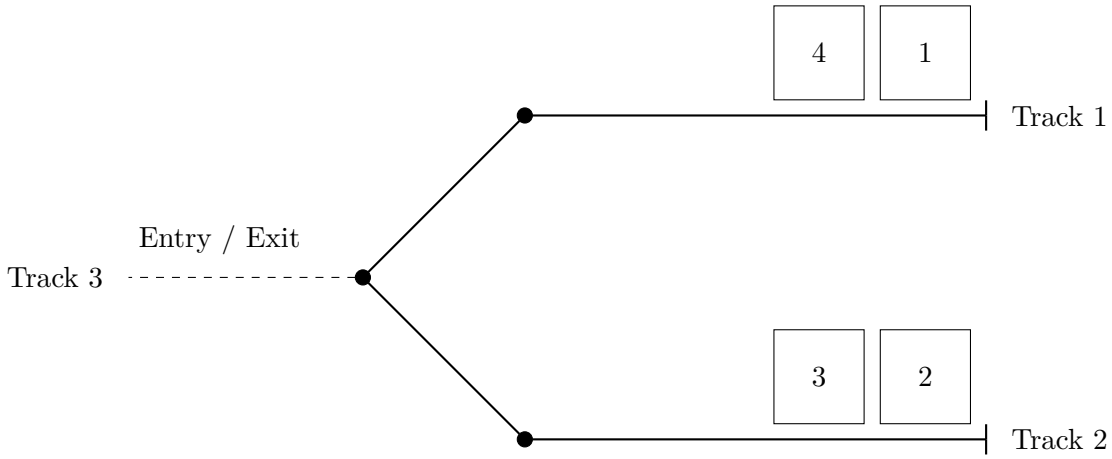

Figure 4-9: The configuration of trains on the tracks after crossover has been performed. Units 1, 2, and 4 are parked at the same track as they were in Parent B, and unit 3 is parked at the same track as it was in Parent A. The order in which they are parked on the track is determined by their arrival order (First-In-Last-Out). All units can now depart in their designated departure order, without being blocked by another train.

This concludes the explanation of CBC, the evolutionary algorithm created in this research. In this chapter, the literature-based components of the EA have been discussed, including the representation and evaluation of solutions, the creation of an initial population, tournament selection as a mechanism for survivor selection, and ways in which CBC is hybridised. Furthermore, innovative contributions of this method have been laid out, including a diversity measure, diversity-based parent selection, and most importantly, a crossover operator including a heuristic that guides the crossover process. The next chapter discusses experiments that compare the performance of CBC against Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA), a more general EA, and HIP, the state-of-the-art method used by the NS.

Chapter 5

Results

Now that the methods have been laid out in the previous chapter, we want to evaluate their performance by running experiments. The goals of these experiments are threefold: the first goal is to obtain insights into the mechanics of the developed methods, which can help in tuning parameters or deciding when an Evolutionary Algorithm (EA) might be useful. The second set of experiments are used to evaluate if an EA can help in day-to-day planning. And the final set of experiments is geared towards the potential of using an EA as an aid in strategic decision making.

An analysis of various components of EAs can help in understanding when and why it works well, and to gain insight into the effect of different design choices of the EA. More specifically, for Conflict-Based Crossover (CBC) it is interesting to know the effect of the diversity-based parent selection and the conflict-based crossover, compared to control methods that use random parent selection and uniform crossover, respectively. For both CBC and Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) it is valuable to assess the effect of the extra local search step at the end of each generation and determine if it has a positive impact on the results. The first set of experiments aim to provide an analysis of these subjects, while also comparing the EAs with Hybrid Integrated Planner (HIP) to get an idea of how their respective performances relate.

The remaining experiments are focused on testing CBC on realistic locations and problem instances and assessing whether the algorithm can be used for day-to-day planning and/or strategic decision-making. The former requires the algorithm to be able to find more feasible solutions than HIP within a small timeframe, whereas the latter has the goal of finding more feasible solutions than HIP given a relatively long runtime.

The chapter is organised as follows. First, several general concepts regarding the setup of experiments are discussed in Section 5-1. Next, Section 5-2 shows results of experiments ran on an artificial location, which provide the basis for an analytical breakdown of the EAs, while simultaneously acting as a proof of concept for CBC. Section 5-3 answers the question whether the results on this artificial location transfer to scenarios on real-world locations. Section 5-4 discusses results of experiments that are run to determine the proficiency of an

EA compared to HIP when it comes to day-to-day planning (Section 5-4-2), and compares the usefulness of HIP and an EA for strategic decision-making (Section 5-4-3). Finally, Section 5-5 offers various hypotheses that aim to explain the results.

5-1 Setup

Before the actual experiments are discussed, various concepts related to the setup of the experiments are discussed. The hardware that is used is described for reproducibility. The different locations that are presented have a significant effect on the results, so for future work it is important to know the different locations that have been tested. The methods act as a reference for the reader. The problem instances that are used are representative of real-world instances, and their composition is also important for the results, which is why they are described here. Termination criteria are important to consider because a short run (i.e. termination criterion is to stop relatively quickly) can have different relative results compared to a long run. And finally, metrics that are used to analyse the results are explained.

5-1-1 Hardware

Because of a hardware change during the research, not all experiments are performed at the same machine. Both machines run Windows 10 (64 bit), however, their specifications differ. The first experiments are run on an HP Zbook 15 with an Intel(R) Core(TM) i7-4700MQ CPU @ 2.40 GHz processor, with 4 cores and 8 logical processors, and 8 GB of RAM.

Later experiments are run on a Lenovo P1 Gen 3 with an Intel(R) Core(TM) i7-10750H CPU @ 2.6 GHz (up to 5.0 GHz) processor, with 6 cores and 12 logical processors, and 16GB of RAM. In the remainder of this chapter, for each experiment the machine it is run on is indicated.

5-1-2 Locations

To test the performance of the different methods a total of three locations, or shunting yards, is used. These are an artificially created test location, de Grote Binckhorst, and de Kleine Binchkorst. Figures 5-1, 5-2, and 5-3 show layouts of these respective locations. The green vertical bars signify the entries and exits of the track, the red vertical bars signify bumpers (the end of the track), the blue and yellow dots represent switches, and the green boxes at Kleine Binckhorst depict service stations.

The NS categorises its shunting yards into one of two categories: shuffleboard locations and carousel locations. The artificial location and Grote Binckhorst are shuffleboard locations, while Kleine Binckhorst is a carousel location. The main difference is that shuffleboard locations are composed mostly of parking tracks with one entry/exit, which means that trains have to depart in a Last-In-First-Out (LIFO) order. Carousel locations, on the contrary, have a lot of parking tracks with two entries/exits, meaning that trains can also leave in a First-In-First-Out (FIFO) order. Generally, this means that trains are able to move around more without being blocked by another train.

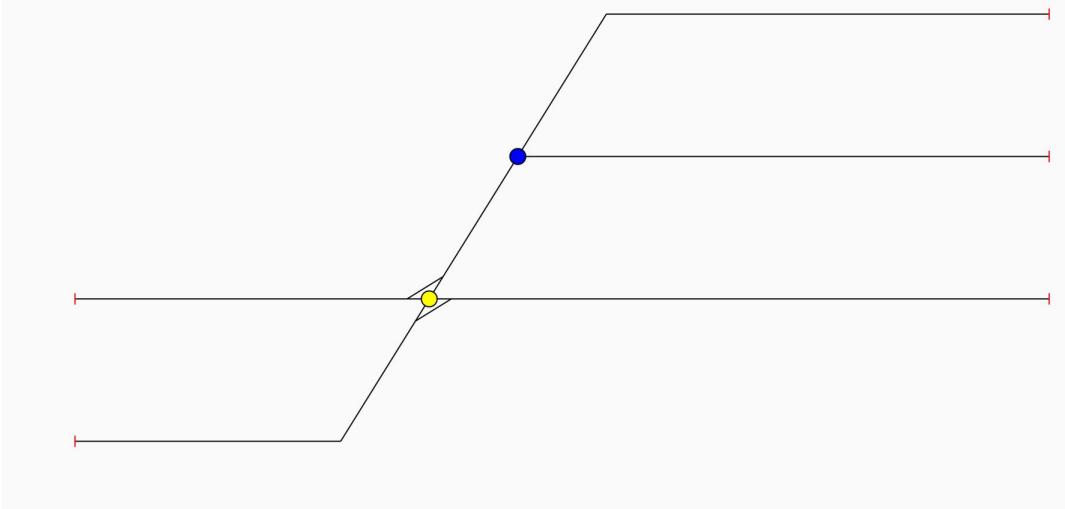

Figure 5-1: Layout of the artificial location.

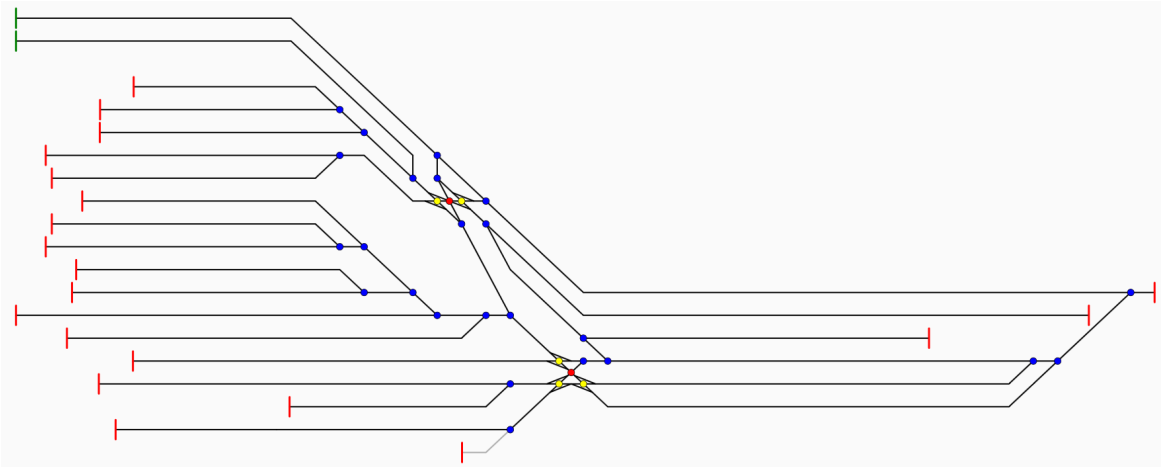

Figure 5-2: Layout of de Grote Binckhorst.

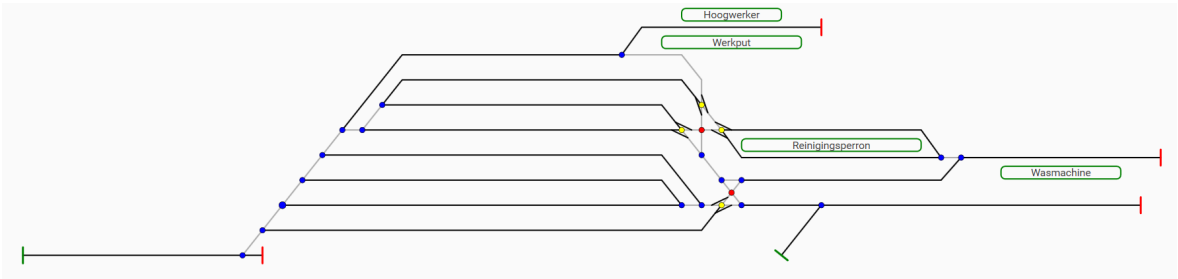

Figure 5-3: Layout of de Kleine Binckhorst.

The initial expectation of the NS was that an EA may provide the most benefits on shuffleboard locations. The argumentation for this is that if a significant portion of trains on such a location is parked on the wrong track, it is very difficult for HIP to adjust this because it requires many changes. An EA is expected to be better at this since large changes are made during crossover.

The first experiments are run on the artificial shuffleboard location. The thought behind this is that it is relatively small and simple, making it easier to analyse results manually. This may help in determining why the EA performs better on some instances than on others. And a shuffleboard location is chosen because it is expected to be the best fit for EAs, as discussed in the previous paragraph.

The other locations are used in later experiments, to see if the results from the test location transfer to the real world. Both a shuffleboard location and a carousel location are tested, to see the difference in performance of EAs on locations with varying characteristics.

5-1-3 Methods

Chapter 4 describes the EA that is developed in this research to solve Train Unit Shunting and Servicing (TUSS), called CBC. In the literature review in Chapter 2 a more general EA called GOMEA is described, which can also be used to solve TUSS, using the crossover operator described in Section 4-4-1. Components of both methods can be varied, and different variants of the EAs are tested to determine which one seems to work best for TUSS. Table 5-1 lists an overview of the different variants that are used, which are denoted by their abbreviation in the remainder of this chapter.

The methods are run with various population sizes. For HIP a population size represents the number of parallel starts with a different random seed. From here on, when a method is followed by a dash and a number, it specifies that method with a certain population size (e.g. CBC-32 means CBC with population size 32).

5-1-4 Problem instances

Each test is performed on a set of problem instances that specify the arrival and departure times of trains and possibly the service tasks that need to be performed. These problem instances are randomly generated for each location, using a probability distribution of arrival and departure times of trains that resembles the arrival and departure times of real-world scenarios. The following settings differ throughout experiments:

- The number of instances that are used. Running the methods on more instances gives a more accurate view of the performance of different methods, but is also more time-consuming.
- The number of train units in a problem instance, referring to all train units that arrive and depart in a scenario. With more train units problems become harder to solve.
- Whether service tasks are included in the scenario. Adding service tasks to problem instances adds additional complexity, so the behaviour of HIP and the EAs might be influenced by this parameter.

Method	Description
CBC	The 'main' EA, with conflict based crossover, diversity-based parent selection, tournament size 2, and HIP search for every individual at the end of each generation.
UC	The same as CBC, except that uniform crossover is used.
CBC-WOH	The same as CBC, except that no HIP search is performed at the end of a generation.
CBC-RPS	The same as CBC, except that random parent selection is used.
CBC-TS4	The same as CBC, except that a tournament size of 4 is used.
GOMEA-U	GOMEA with univariate grouping (every group consists of a single train unit). Includes HIP search for every individual at the end of every generation.
GOMEA-R	The same as GOMEA-U, except that the trains are randomly divided into 5 groups of (almost) equal size.
HIP	The local search method used by the NS, which acts as a benchmark for the EA.

Table 5-1: Overview of algorithm abbreviations.

5-1-5 Termination criteria

Each run of an algorithm needs to have termination criteria to prevent it from running forever and to enable a fair comparison by giving different methods the same termination criteria. One criterion that is used by default for all methods is finding a feasible solution. This is the goal of all the methods, so when this is achieved the algorithm can stop.

Other termination criteria that are used are the number of function evaluations and the total runtime. The latter simply stops when the specified maximum time has passed since the start of the algorithm. The former sums up all function evaluations and stops as soon as the maximum is reached. For HIP, this means that each start gets an equal portion of the total number of evaluations (e.g. if HIP is started with 100,000 max evaluations and 5 parallel runs, each run is limited to 20,000 function evaluations).

5-1-6 Metrics

To interpret the results of experiments, a multitude of data measures is used to give insight into the data. These metrics are calculated per method and population size and are averaged over all problem instances an experiment is run on, as will be apparent from the results displayed in this chapter. Some metrics are split into *solved* and *unsolved*, meaning the averages are calculated separately for the instances where a method did find a feasible solution and the instances where it did not. The following measures are used:

- **Solved %**: the percentage of problem instances solved. Arguably the most important metric, because the NS wants a method that finds feasible solutions when they exist.
- **Avg. time (sec)**: the average time it took a method to complete. This is split into solved and unsolved. Solved indicates how quick a method converges to a solution,

while unsolved highlights the difference in runtimes between methods, given the same number of function evaluations. The latter is only relevant for runs where runtime is not a termination criterion.

- **Avg. crossovers**: displays how many crossovers a method uses in one run on average. This is split into solved and unsolved. It can show differences in the number of crossovers used by different EAs and this information can possibly be used to conclude something about the number of crossovers that maximises performance of an EA for TUSS.
- **Evaluations per CO**: displays the average ratio of the total number of function evaluations to crossovers. The majority of function evaluations come from local search in the EAs, and as such this metric is an indicator of the balance between crossovers and local search within the EAs.
- **CO Improvement (%)**: a measure that calculates the percentage of times in which a crossover yields an improvement; this is calculated before local search is applied to the new solution, and after, to assess the impact of local search on the quality of new solutions created with crossover. Furthermore, this provides a way to compare different crossover methods with each other.

5-2 Proof of concept

In the previous chapters two EAs have been laid out that can be used to solve TUSS, namely GOMEA and CBC. The first experiments that are run compare different variants of these methods with each other, to see which method holds the most promise. Furthermore, the results are compared with HIP to determine if any EA may be able to outperform a state-of-the-art local search method.

Hardware	HP ZBook 15	**Location**	Artificial location
Methods	All	**Problem instances**	50
Evaluation limit	1,600,000	**Train count**	21
Time limit (min)	None	**Service tasks**	No

Table 5-2: Setup of first experiment.

Table 5-2 shows the setup for this test. All algorithms that are listed in Table 5-1 are tested on the artificial location, which is chosen because it may be easier to analyse the behaviour of different methods on this shunting yard because it is relatively simple, as is explained in Section 5-1-2. The algorithms are run on 50 problem instances because this provides sufficient data to enable a proper analysis. Each problem instance has 21 train units; this is the maximum number of train units that fit on this location. No service tasks are included for simplicity: if the EAs do not perform well on problem instances without service tasks, they will likely also not perform well on problem instances that do include service tasks. The evaluation limit is set at 1,600,000 evaluations and no time limit is set. This seemed like the best way to compare the impact of different methods on the results because every method can now make a total of 1,600,000 different plans and evaluate these. The number 1,600,000 is chosen because preliminary experiments showed that this takes up to 5 minutes for these

scenarios, which is relatively short and a timespan within which the algorithms could be used for day-to-day planning. Finally, different population sizes are used for the various methods to judge their impact as well.

5-2-1 Results

Table 5-3 shows the results of the experiments. The results for the different metrics in the table are discussed in this section.[1] The most important takeaway of these results is that CBC is the best performing method in these experiments, even outperforming HIP, which is elaborated on now.

Solved (%). The most important measure is the number of instances that are solved by a method within the allowed number of evaluations. From the table, we can see that the most successful method is CBC with a population size of 32, solving 86% of instances, which amounts to solving 43 of 50 instances. Since 46 were solved by at least one method, it means 3 instances were not solved by CBC-32 but were solved by another method. This implies that although CBC is generally the best performing method for these problem instances, there are cases where a different method is preferred, implying that some sort of automatic algorithm selection process might benefit the overall performance of the algorithm ecosystem for TUSS.

The results show that CBC is the best method among the EAs. It clearly outperforms GOMEA and also outperforms the runs where one setting within CBC was changed. CBC with a tournament size of 4 was very close in performance though, so this parameter might be worth investigating more in the future. There are two likely reasons for the poor performance of GOMEA. First, the crossover in GOMEA does not use any domain knowledge and therefore does not necessarily steer the search towards promising parts of the search space, thereby reducing the effectiveness of crossover. Second, because of the structure of GOMEA it has significantly fewer evaluations per crossover; since crossover is a disruptive operation, it is likely that solutions get disrupted too often and thus not enabling local search the opportunity to move to a feasible solution.

HIP-8 (the most successful of the different HIP runs) solved 78% of instances. There are 5 instances where CBC-32 finds a solution and HIP-8 does not, and one instance where the reverse is true. Concluding, we can state that in this experiment the best EA slightly outperforms the benchmark, and GOMEA seems to be an unviable option for TUSS. Therefore, in the remainder of this chapter the focus is mostly on CBC and HIP.

[1] The runtimes for HIP with 16 and 32 starts is missing due to a corruption of the data. However, these can be assumed to be slightly longer than the runtime for HIP with 8 starts.

Method	Pop. size	Solved (%)	Avg. time (sec)		Avg. crossovers		Evaluations per CO	CO improvement (%)	
			Solved	*Unsolved*	*Solved*	*Unsolved*		*Before LS*	*After LS*
CBC	8	80	**57**	305	**131**	881	1996	3	17
	16	78	73	287	154	695	2472	3	28
	32	**86**	103	286	199	608	2850	2	41
UC	8	78	66	297	139	739	2342	0	18
	16	74	88	282	173	607	2769	0	25
	32	74	124	284	227	**569**	**2975**	0	32
CBC-WOH	32	72	94	310	433	2203	884	2	27
CBC-RPS	32	72	79	299	156	729	2404	3	34
CBC-TS4	32	84	98	317	189	704	2538	2	27
GOMEA-U	8	76	133	321	1288	3117	519	14	42
	16	48	151	312	1476	3028	529	15	48
	32	52	160	307	1550	2960	542	**17**	**55**
GOMEA-R	8	70	83	302	455	1785	919	2	28
	16	68	87	290	478	1659	978	2	36
	32	70	107	289	574	1548	1043	2	46
HIP	8	78	78	**281**	-	-	-	-	-
	16	74	-	-	-	-	-	-	-
	32	72	-	-	-	-	-	-	-

Table 5-3: Results for the various methods. In each column, the number in bold is the best result for that metric. Of the 50 problem instances, 46 are solved by at least one of the methods, meaning that at least 92% of instances is feasible.

Average time. The average time shows the average runtime of the different methods on the 50 problem instances and is split into the runtime for solved and unsolved instances. Most interesting here is to look at the time for the unsolved instances, to determine how big the overhead is that the EAs have in terms of runtime compared to HIP. The highest runtime that any of the CBC variants has for unsolved instances is 317 seconds, which is a 13% increase compared to the runtime of HIP. This is a small difference and implies that CBC is not too slow for day-to-day planning.

EA analytics. The remaining columns give more insights into the behaviour of the EAs. The most important insights that are gained from these results are the following:

- As can be seen from the average crossovers for solved and unsolved instances, GOMEA has a lot more crossovers than CBC (which is expected from the way GOMEA is set up). Because crossovers are highly disruptive to solutions (a crossover makes large changes), doing this too often likely introduces more conflicts than it resolves. As mentioned above, this may be a reason why GOMEA's performance is subpar.
- Closely correlated to the previous point is the number of evaluations per crossover. These are higher for the CBC variants (except CBC-WOH), and as such the balance between crossover and local search is tilted more towards local search in these methods compared to the GOMEA methods.
- The CO improvement columns show the percentage of crossovers that yielded an improvement of the solution; before the application of local search to the new solution, and after. It shows that, before local search, especially for CBC and its variants, the solutions that are created from crossover rarely improve over their parents. After running local search on these solutions, however, up to 41% of new solutions (for CBC-32) are an improvement over their parents. This signifies the immense importance of augmenting the EAs for TUSS with local search.

5-2-2 The HIP effect

All the evolutionary algorithms, except CBC-WOH, run HIP's Variable Neighbourhood Search (VNS) for 4000 function evaluations on each individual at the end of each generation. To test the impact of this extra step in the EAs two things are measured.

First, CBC is run without this extra step at the end of each generation (CBC-WOH), the results of which are in Table 5-3. CBC-WOH solves 72% of instances, which is 14% less than the 86% of CBC-32. This indicates that VNS has a positive impact on the performance of CBC.

Second, a couple of statistics relating to the HIP search are logged and displayed in Table 5-4. The HIP searches column shows the average number of times the extra search step is started in a single run. The last two columns show how often HIP search finds an improvement as a percentage of the total number of searches and how often it finds the final solution as a percentage of the total number of solutions found, respectively. The last column shows that this extra search step finds the feasible solution from 59% to 84% of cases. This implies that

Method	Pop. size	HIP searches	HIP improvements (%)	HIP found solution (%)
CBC	8	278	6	65
	16	268	11	64
	32	247	19	67
UC	8	268	10	59
	16	283	17	84
	32	309	22	76

Table 5-4: This table illustrates the importance of including HIP search in the EA. We can see that for CBC with a population size of 32 (the best performing algorithm), in 67% of the solved instances, the solution was found by the HIP search at the end of the generation.

it is a vital step in the algorithm, and it supports the notion that local search is better at the exploitation of good solutions (i.e. finding the optimum when the current solution is already close), especially when it allows for a temporary move away from a local optimum, as VNS does.

5-3 Real-world locations

The previous section laid out results of all different methods on a test location. These results provided insights into the mechanics of the EAs and showed which methods performed best. From now on all comparisons are between HIP (the benchmark) and CBC, justified by the results in Table 5-3 which showed that CBC was the best performing EA. The following experiments are run on two real-world locations to determine if the performance of CBC on a test location also transfers to locations that are used in practice. A carousel location (Kleine Binckhorst) and a shuffleboard location (Grote Binckhorst) are tested (both are discussed in Section 5-1-2) to evaluate the performance difference on locations with different characteristics.

5-3-1 Carousel location

Table 5-5 shows the setup of the first experiment run on a carousel location. Each problem instance consists of 29 train units because preliminary experiments showed this to be an amount which had the right balance between being too simple (and thus both methods solving all problems) or too hard (and both methods solving nothing). At this point a hardware switch was made to the Lenovo P1, so results of this experiment cannot be compared with previous experiments.

Hardware	Lenovo P1	**Location**	Kleine Binckhorst
Methods	HIP, CBC	**Problem instances**	100
Evaluation limit	1,600,000	**Train count**	29
Time limit (min)	None	**Service tasks**	No

Table 5-5: Setup of experiment 2.

Table 5-6 shows the results of this experiment. Both CBC and HIP are run with various population sizes / different starts to get an indication of the optimal setting for both methods. The results show that for both methods a size of 8 works best (for HIP this is tied with 4 starts). For CBC the performance deteriorates as the population size increases, and for HIP this is also the case from 16 starts downwards. The likely reason for this for HIP is that with more starts, each start has fewer evaluations at its disposal and thus does not have enough time to reach a feasible solution. A similar argument can be made for CBC: with a larger population, the total number of evaluations has to be split over more individuals, and thus each individual does not get enough time to converge.

Method	Pop. size	Solved (%)	Avg. time (sec)	
			Solved	*Unsolved*
CBC	8	**99**	266	844
	16	97	389	821
	32	69	535	766
	64	20	803	1002
HIP	1	40	393	1110
	2	59	273	668
	4	62	**233**	**491**
	8	62	312	524
	16	49	-	-
	32	30	-	-

Table 5-6: Results for Kleine Binckhorst with 29 trains.

The results show that the best performance of CBC is markedly better than the best performance of HIP: a gap of 37%. The NS considers a shunting yard to have at least a capacity of x if it can find solutions to problems with x trains in at least 95% of cases. In this particular experiment, that means that the results of HIP would deem the solving rate insufficient to claim a capacity of 29 trains for these problems on Kleine Binckhorst, whereas the solving rate of CBC is sufficient to claim this capacity.

One caveat is the runtime. Whereas the experiments in Section 5-2 showed a small difference in runtime between HIP and CBC (less than 15%) the difference on these problem instances is bigger. CBC-8 takes 62% longer on its unsolved instance than the 524 seconds that HIP-8 takes on its unsolved instances. This additional runtime needed by CBC to reach the same number of evaluations is likely caused by several factors. First, crossover is a large operation that adds some runtime overhead. Second, in each generation all individuals have to wait until the last individual finishes its local search (since they run in parallel, but all individuals need to be done with local search before offspring selection can be performed). The different starts in HIP (also running in parallel) do not have to wait on one another, giving HIP a runtime advantage.

In an experimental setting this additional runtime is not a problem, because it shows the positive impact of crossover on the search process. However, in a practical setting it makes the comparison biased in favour of CBC, because in a practical setting what matters is how

long it takes before the algorithm completes, not how many evaluations it has at its disposal. Later experiments will address this by allowing both methods the same runtime.

5-3-2 Shuffleboard location

The previous section shows promising results for CBC on a carousel location. This section describes results from running CBC and HIP on a shuffleboard location. Table 5-7 shows the setup of these experiments, which is similar to the previous experiment. Again, the number of train units is chosen because preliminary experiments showed it to be the right count to yield interesting results. Because both CBC and HIP showed the best results with a population size/start size of 8, this is the size that is used from here onwards. Due to time constraints, it is not possible to test multiple different sizes for every experiment.

Hardware	Lenovo P1	**Location**	Grote Binckhorst
Methods	HIP-8, CBC-8	**Problem instances**	100
Evaluation limit	1,600,000	**Train count**	48
Time limit (min)	None	**Service tasks**	No

Table 5-7: Setup of experiment 3.

Table 5-8 shows the results of these runs, which are surprising given the previous results on the artificial location and Kleine Binckhorst. In these previous tests CBC outperformed HIP; however, on de Grote Binckhorst the performance of CBC is nowhere near the performance of HIP. To assess if the addition of service tasks to the problem instances has an impact on the relative performance of CBC and HIP, experiments with service tasks are also run on Grote Binckhorst. These results can be found in Appendix C and are similar to the results displayed here: HIP evidently outperforms CBC.

Method	Pop. size	Solved (%)	Avg. time (sec)	
			Solved	*Unsolved*
CBC	8	10	721	1067
HIP	8	100	283	-

Table 5-8: Results for Grote Binckhorst with 48 trains and no service tasks.

Based on these results it seems that CBC as it stands currently has little promise on shuffleboard locations. Section 5-5 offers various hypotheses that aim to explain the difference in performance of CBC on Grote Binckhorst and Kleine Binckhorst. Because Kleine Binckhorst shows more promise for CBC, the remainder of the experiments are performed on this location. The next section discusses these final experiments, which include service tasks.

5-4 CBC as a practical tool

The initial results for CBC on Kleine Binckhorst are promising and warrant further exploration of CBC on this location. These final experiments include service tasks, and as such all

subproblems of TUSS (i.e. matching, parking, servicing and routing) are now included. The goals of these experiments are threefold. The first goal is to assess the impact of crossover on the search process. The second and third goals are to determine whether CBC can be useful in day-to-day planning and in strategic decision-making, respectively. The tests consist of three parts corresponding to these goals:

1. First, a comparison between HIP and CBC is made showing the impact of crossover on the search process; this is done in a similar fashion as previously, by giving both methods the same number of function evaluations.
2. Then, the possibility of using CBC for day-to-day planning is examined, by comparing it with HIP given the same (relatively short) runtime.
3. Finally, the potential for CBC as an aid in strategic decision-making is explored. This is done by giving both HIP and CBC a plethora of runtime to see if this enables them to find solutions for difficult problems.

5-4-1 Analytical comparison

The following runs compare HIP and CBC given the same number of function evaluations. This shows the impact of crossover on the search process since that is the main difference between local search and an EA. They are run on 100 problem instances of varying sizes (from 18 trains to 23) to see how both methods perform for problems increasing in difficulty. Table 5-9 shows an overview of the settings for this experiment.

Hardware	Lenovo P1	**Location**	Kleine Binckhorst
Methods	HIP-8, CBC-8	**Problem instances**	100
Evaluation limit	1,600,000	**Train count**	18-23
Time limit (min)	None	**Service tasks**	Yes

Table 5-9: Setup of experiment 4.

Figure 5-4 shows results for the various sizes of the scenarios. It is evident from these results that given an equal evaluation budget, CBC-8 outperforms HIP-8, and there are rarely cases where HIP does find a solution and CBC does not (indicated by the marginal difference between CBC-8 and the total number of instances solved by either of the methods). This implies that CBC offers an effective way of exploring the search space, whereas HIP seems to have more difficulty getting out of local optima.

However, Figure 5-5 shows a comparison of the runtimes of CBC-8 and HIP-8. CBC takes significantly more time than HIP and thus part of the success of the EA can be attributed to the fact that it takes more time to find a solution. To make a comparison that is deemed fairer, the next section compares both methods given the same runtime.

5-4-2 Day-to-day planning

The previous section shows that given the same evaluation budget, CBC-8 solves significantly more scenarios than HIP-8 on Kleine Binckhorst, regardless of the number of train units in

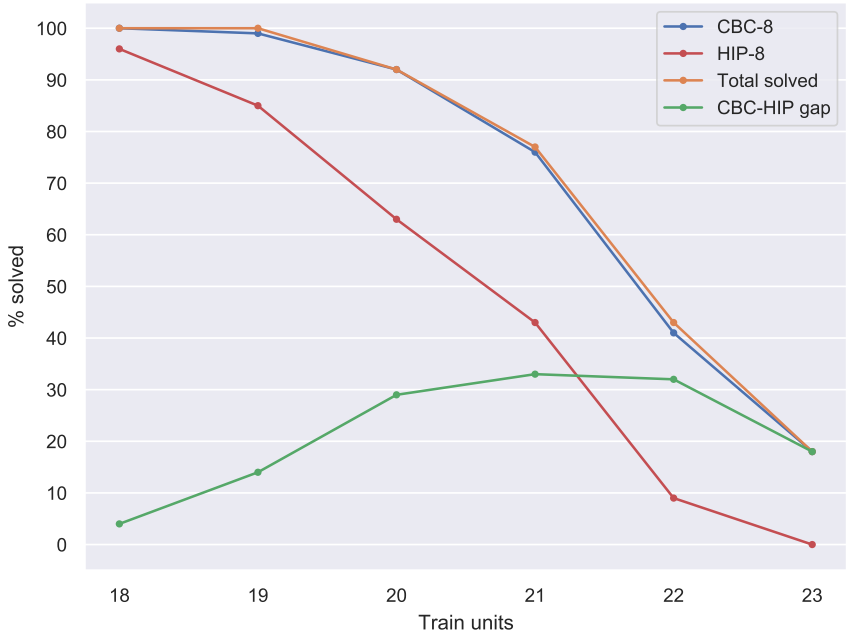

Figure 5-4: The percentage of instances solved by CBC-8 and HIP-8 for various scenario sizes. The orange line shows the total percentage of instances solved (by either of the methods) and the green line shows the difference between CBC and HIP.

the scenario (given the scenarios are not too easy or too difficult for both methods). However, in a practical setting the comparison is unfair because the runtime of CBC is much higher. When investigating the possibility of using CBC as an aid in day-to-day planning, runtime is a more suitable termination criterion, since human planners wish to receive possible plans within a short timeframe. The next tests compare HIP and CBC for this purpose.

Hardware	Lenovo P1	**Location**	Kleine Binckhorst
Methods	HIP-8, CBC-8	**Problem instances**	100
Evaluation limit	None	**Train count**	20
Time limit (min)	2, 4, 6, 8, 10	**Service tasks**	Yes

Table 5-10: Setup of experiment 5.

Table 5-10 shows the setup for this experiment. HIP and CBC are run for various (maximum) runtimes, and the results of these runs are depicted in Figure 5-6. These results show that the gap between the two methods is much smaller given the same runtime. However, CBC still outperforms HIP for all runtimes but 2 minutes, where their performance is equal. This implies that CBC is a viable option for aiding day-to-day planning.

5-4-3 Strategic decision-making

Finally, the possibility of using CBC as a tool to guide the strategic decision-making process is examined. Strategic decisions involve determining the capacity of a shunting yard; this, in turn, steers the decision whether a shunting yard should be expanded or not. To decide the

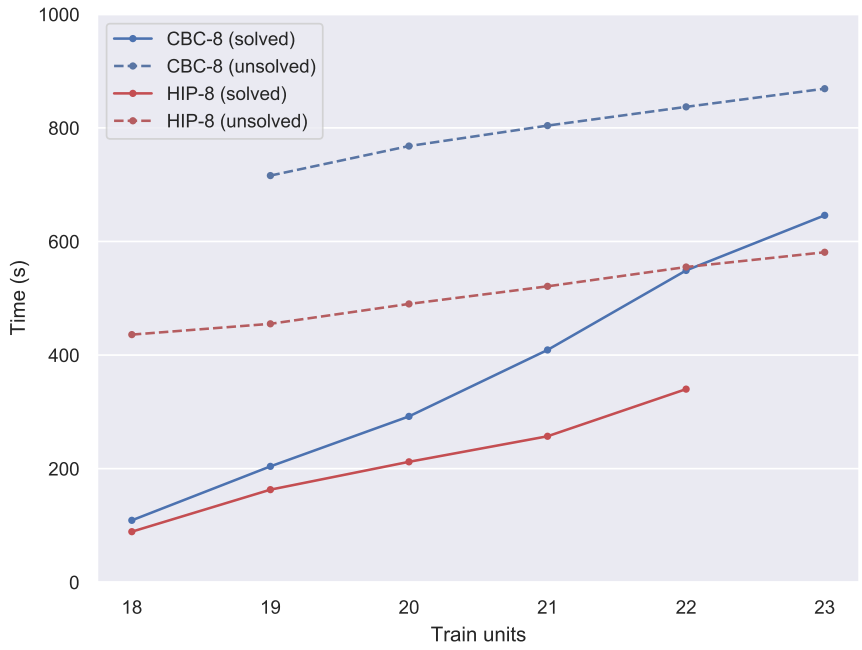

Figure 5-5: The average runtimes of CBC-8 and HIP-8, split in runtimes for solved and unsolved instances.

capacity the NS needs to know for what scenario size feasible solutions can still be found. 95% of instances for a particular scenario size (i.e. the number of train units) should be solvable for the NS to decide that a yard has at least that capacity. To determine this, algorithms can run longer than for day-to-day planning, since a solution is not needed immediately.

Hardware	Lenovo P1	**Location**	Kleine Binckhorst
Methods	HIP-8, CBC-8	**Problem instances**	23
Evaluation limit	None	**Train count**	21
Time limit (min)	240	**Service tasks**	Yes

Table 5-11: Setup of experiment 6.

The last experiment compares how well CBC and HIP can find solutions for very difficult problems, given a long runtime. Figure 5-4 shows that for scenarios with 21 train units, 77 out of 100 problems were solved. In a practical setting it would be interesting for the NS to know if the other 23 instances can also be solved, given a longer runtime. If they can, the NS can argue that the capacity of Kleine Binckhorst for these particular scenarios is at least 21 train units. Therefore, both methods are run on these 23 unsolved instances with a maximum runtime of 4 hours.

Table 5-12 shows the results. From the 23 instances, 22 were solved by at least one of the two methods. This indicates that the capacity of Kleine Binckhorst is at least 21 train units for this scenario. CBC solved 87% of instances, amounting to 20 solved out of 23. HIP only solves 52%, amounting to 12 out of 23 instances solved. CBC on average also takes less time to get to the feasible solutions. These results seem to confirm the intuitive thought that an

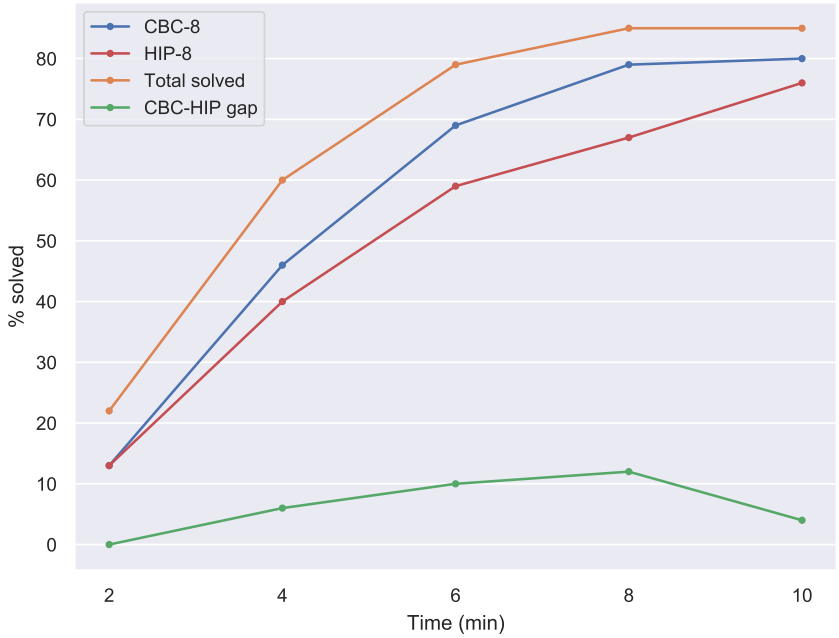

Figure 5-6: The performance of CBC-8 and HIP-8 for various runtimes.

Method	Pop. size	Solved (%)	Avg. time (sec)	
			Solved	*Unsolved*
CBC	8	87	1098	14401
HIP	8	52	2308	14399

Table 5-12: Results for long runs. 22 instances got solved by either of the methods.

evolutionary algorithm does a better job at covering a large part of the search space, while local search is more likely to get stuck in a local optimum.

One method which might improve the results for HIP is to use a lot more random restarts. Instead of using 8 starts and giving them all 4 hours, every start could be terminated after 15 minutes and replaced by a new run starting from a different initial solution. This way the local search could explore a bigger part of the search space. However, the initial solutions start at random positions which makes it unlikely that such a strategy will outperform the EA since the mechanisms of the EA are similar to restarting local search many times. The difference, though, is that the EA does not start the local search from random positions, since the crossover guides it into promising areas of the search space. Another reason why the EA is preferred over a lot of restarts of HIP is that the latter method would require a lot of tuning to determine a good balance between restarts and exploitation of current solutions, whereas the EA works out of the box.

Location	Servicing	Trains	Solved (%)		Avg. crossovers	CO improvement
			CBC-8	*HIP-8*	*Unsolved*	*After LS*
Kleine Binckhorst	Yes	18	100	96	-	42
	Yes	19	99	85	304	35
	Yes	20	92	63	294	32
	Yes	21	76	43	275	28
	Yes	22	41	9	273	27
	Yes	23	18	0	267	27
	No	29	99	62	456	36
Grote Binckhorst	Yes	41	55	100	153	15
	Yes	45	27	100	154	17
	No	48	10	100	188	22

Table 5-13: Comparison of results on Kleine Binckhorst and Grote Binckhorst.

5-5 Discussion of varying results

The results in Section 5-3 show a significant difference in the relative performance of CBC compared to HIP on Kleine Binckhorst and Grote Binckhorst. The 'Solved (%)' column in Table 5-13 illustrates this difference. The final section of this chapter offers hypotheses that aim to explain the varying performance of CBC, using additional data from the results.

The column with the CO improvement in Table 5-13 shows how often crossover improves the solution (after applying hill-climbing local search to the new solution) and clearly shows that the crossovers on Kleine Binckhorst instances are more effective than those on Grote Binckhorst instances. A possible reason can be related to the total amount of parking space occupied. Because Kleine Binckhorst is a carousel location, there is more room for moving trains around while they are at the shunting yard. This yields more flexibility but also decreases the total number of trains that can occupy parking space at the yard. A shuffleboard location such as Grote Binckhorst has a simpler layout and not much flexibility in moving trains around at the yard. Therefore, most of the parking space is occupied. So unlike carousel locations, there are no large unoccupied parts of the yard that may be utilised efficiently by another solution and thereby would benefit from crossover with this other solution. Therefore, we hypothesise that smaller changes to existing solutions fare better on shuffleboard locations because large changes (i.e. crossover) only introduce new conflicts that need to be resolved. And consequently, CBC seems to be a better fit for complex locations that have a larger search space.

Another possible reason for the subpar performance of CBC on Grote Binckhorst can be the total number of trains in the problem instances. These are significantly higher than on Kleine Binckhorst. This makes the probability of introducing conflicts in crossover higher, and as such the local search performed after crossover needs more iterations to resolve these conflicts again. Therefore less time remains to improve the solution. This notion is supported by the average number of crossovers for unsolved instances in Table 5-13. This shows that the average number of crossovers is higher for instances on Kleine Binckhorst, indicating that the

EA needs fewer local search iterations in these instances to get to a local optimum.[2]

Both hypotheses above need to be confirmed by running more experiments. Obvious experiments include comparing CBC and HIP on different carousel and shuffleboard locations; ideally, a shuffleboard location with a smaller capacity to assess the impact of the instance sizes. Another interesting option would be to test CBC and HIP on a less complex shuffleboard location, to see how the complexity of the location impacts the results.

We conclude the chapter by looking back at the goals that were set out at the start of these experiments. The first goal was to obtain insights regarding the different EAs. GOMEA turned out not to be a viable option, since it got outperformed by HIP and CBC. CBC on the other hand is a viable option for TUSS, and its conflict-based crossover and diversity-based parent selection both seem to have a positive impact on performance. The former steers the search towards favourable parts of the search space, while the latter ensures that the algorithm does not converge prematurely.

Other goals of the experiments were to assess the usefulness of an EA in a real-world setting, for both day-to-day planning and strategic decision-making. The first notion is that CBC currently does not seem to be a realistic option for shuffleboard locations because of poor performance. However, on carousel locations it showed positive results. It solved more instances than HIP given the same short runtime, indicating CBC could be an improvement over HIP for day-to-day planning. And it solved a lot more instances than HIP given the same long runtime, indicating that CBC could also be an improvement over HIP for use in strategic decision-making.

The last aim of the experiments was to hopefully gain some insights into the mechanisms of the EA, and why it does (not) perform well. This last section offers some hypotheses for potential reasons why CBC performs better on a carousel location than on a shuffleboard location; more experiments are needed to confirm these hypotheses. The next chapter concludes this thesis and provides pointers for future research.

[2]The local search after crossover runs until no improvement can be found. Thus more crossovers means less local search iterations after each crossover, meaning a local optimum is found quicker

Chapter 6

Conclusion

The main goal of this research is to investigate if an Evolutionary Algorithm (EA) can be developed for Train Unit Shunting and Servicing (TUSS) and if and when that EA can outperform a current state-of-the-art heuristic called Hybrid Integrated Planner (HIP). The previous chapters of this thesis describe the workings of the EA created and present a discussion of the results of experiments that are run. This chapter concludes the research, to evaluate how well the research questions, that were set out at the beginning of this thesis, are answered. Furthermore, some pointers for future research are given.

To develop an EA for TUSS a crossover operator is needed that is able to mix two solutions to form a new, valid solution. To yield positive results the operator should also be effective; i.e. it should mix solutions in such a way that the new solution has a high probability of being an improvement. Conflict-based crossover, presented in Section 4-4-2, offers a way to do this. It is a crossover operator which uses heuristics that are devised from domain knowledge to provide an efficient search through the search space. The results in Section 5-2-1 imply that this crossover is effective since it outperforms an identical EA that uses uniform crossover instead. Altogether, the tentative conclusion can be drawn that conflict-based crossover as it is presented in this thesis is a useful crossover operator for TUSS, thereby answering RQ 1.

Other components of Conflict-Based Crossover (CBC), the EA designed in this research which is named after its crossover operator, that stand out are the diversity-based parent selection and the high level of hybridisation, with local search performed right after crossover and at the end of every generation (after offspring has been selected). The analysis of the results in Sections 5-2-1 and 5-2-2 shows that both the diversity-based parent selection and the local search improve the effectiveness of the EA.

To answer RQ 2 and assess whether CBC can be used in practice as a day-to-day planning aid, it was tested on real-world locations against HIP, where both methods are given the same (short) runtime. On a shuffleboard location HIP unquestionably outperformed the EA. On a carousel location the EA outperformed HIP: given various short runtimes ranging from 2 to 10 minutes, CBC solved as many or more instances than HIP, as described in Section 5-4-2. This implies that CBC is a viable option for use in day-to-day planning, for at least a subset of locations.

Another application of these algorithms lies in strategic decision making, where they can help in deciding the capacity of a shunting yard. To answer RQ 3 and evaluate the effectiveness of CBC (compared to HIP) on long runs, both methods were run on a set of instances that were considered difficult, with a maximum runtime of 4 hours, to see if they would eventually find a solution. CBC outperformed HIP here: of the 23 instances, there were 2 instances where HIP found a solution and CBC did not, and there were 10 instances where the reverse was true. This shows good promise for CBC in aiding the strategic decision-making process.

The last research question (RQ 4) is aimed at knowing when and why an EA outperforms local search for TUSS. The experiments show that CBC does perform well on Kleine Binckhorst, a carousel location, and does not perform well on Grote Binckhorst, a shuffleboard location. Section 5-5 offers two hypotheses for the varying performance of CBC:

1. An EA is not well-suited for shuffleboard locations because a larger portion of the yard is occupied by trains, which makes crossover less effective. Because of their simple layout, shuffleboard locations also have a smaller search space, which benefits local search. The opposite is true for carousel locations: these have a larger search space, and as such an EA holds more promise on these.
2. Another reason for the difference in performance can be related to the number of trains in the instances. The problem instances for Grote Binckhorst have significantly more trains than the instances on Kleine Binckhorst. Because of this, crossover has a higher probability of introducing conflicts and becomes less effective.

Because these are only two locations, it is premature to conclude that CBC does perform well on all carousel locations and does not perform well on all shuffleboard locations, or similarly that the number of trains in problem instances is a deciding factor. Experiments on different locations of different sizes are warranted to test the hypotheses above.

In conclusion, results for CBC are promising and warrant further investigation into this method. Its performance on a shuffleboard location was subpar compared to HIP, but it excelled on a carousel location. On this location it showed good performance for day-to-day planning as well as strategic decision making, compared to a state-of-the-art benchmark method. Therefore we think it is worthwhile to further research CBC and the next section discusses some pointers for this future research.

6-1 Future work

There are several interesting areas related to CBC and TUSS that can be researched in future work. These consist of both practical (relevant to the NS) and theoretical (relevant from a research perspective) topics. The most important ones are discussed here.

6-1-1 Splits and combines

From a practical point of view, one important aspect of TUSS for use in a real-world setting is currently missing in CBC, which is trains that consist of multiple units and the splits and combines that accompany this additional dimension. This has not been added to the EA

since it is not trivial to implement and there is limited time; the difficulty lies in adapting the crossover to be able to handle splits and combines. However, it is possible and Appendix D shows how the crossover operator introduced in Section 4-4-1 can be adjusted to do so. Future research could then focus on implementing this theory, and evaluating how the addition of splits and combines influences the results of CBC.

6-1-2 Algorithm selection

Experiments showed that CBC performs better than HIP on a carousel location, but the reverse is true on a shuffleboard location, indicating that CBC, as it currently stands, is only relevant for carousel locations. It would be useful to find out the exact reason that the performance of CBC is subpar on shuffleboard locations: some pointers are given in Section 5-5. More research is needed to confirm these hypotheses. This might also give clues on how to improve it in this area.

However, an improvement may not be possible. In that case a useful feature would be to have automatic algorithm selection [63]: based on the problem instance, an algorithm (HIP or CBC in this case) is selected that is the best fit for that particular problem. Such an algorithm selection can be simple: if the location is of the shuffleboard type, use HIP, else, use CBC. But in the tests that were run on the carousel locations, there were also occasions where HIP found a solution and CBC did not. So in hindsight, HIP was a better choice for these instances. If a measure can be found that can estimate reasonably well when that is the case, that could improve the efficiency of the overall ecosystem of algorithms for TUSS.

6-1-3 Implement CBC for open shop scheduling

From a research perspective it is interesting to determine if the insights/methods that are gained/developed in this research can be ported to other (more general) methods from the literature. The Shunt Train Activity Graph (STAG) that is used to represent solutions is similar to a solution representation that can be used for the open shop scheduling problem [64]. Appendix E details how the method developed for TUSS can be applied to open shop scheduling. Future research could focus on extending and implementing this method and assessing its effectiveness for open shop scheduling.

Bibliography

[1] International Union of Railways, "Annual report," tech. rep., 2015.

[2] "NS Jaarcijfers 2019: Treinreiziger gaat er in 5 jaar fors op vooruit." https://nieuws.ns.nl/ns-jaarcijfers-2019--treinreiziger-gaat-er-in-5-jaar-fors-op-vooruit. Accessed: 2021-03-19.

[3] "'NS verhoogt aantal inzetbare treinen in vijf jaar met 37 procent'." https://www.nu.nl/binnenland/4495818/ns-verhoogt-aantal-inzetbare-treinen-in-vijf-jaar-met-37-procent.html. Accessed: 2021-03-19.

[4] R. van den Broek, H. Hoogeveen, M. van den Akker, and B. Huisman, "A local search algorithm for train unit shunting with service scheduling," Jan. 2021.

[5] A. A. Rahmani Hosseinabadi, J. Vahidi, B. Saemi, A. K. Sangaiah, and M. Elhoseny, "Extended genetic algorithm for solving open-shop scheduling problem," *Soft Computing*, vol. 23, p. 5099–5116, July 2019.

[6] D. Bai, Z. H. Zhang, and Q. Zhang, "Flexible open shop scheduling problem to minimize makespan," *Computers and Operations Research*, vol. 67, p. 207–215, Mar. 2016.

[7] G. Kaizhou, C. Zhiguang, Z. Le, C. Zhenghua, H. Yuyan, and P. Quanke, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," 2019.

[8] T. C. Chiang and H. J. Lin, "A simple and effective evolutionary algorithm for multi-objective flexible job shop scheduling," *International Journal of Production Economics*, vol. 141, no. 1, pp. 87–98, 2013.

[9] R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman, "Shunting of passenger train units in a railway station," *Transportation Science*, vol. 39, pp. 261–272, May 2005.

[10] R. M. Lentink, P.-J. Fioole, L. G. Kroon, and C. van't Woudt, "Applying operations research techniques to planning of train shunting," in *Planning in Intelligent Systems*, pp. 415–436, John Wiley & Sons, Inc., Feb. 2006.

[11] L. G. Kroon, R. M. Lentink, and A. Schrijver, "Shunting of passenger train units: An integrated approach," *Transportation Science*, vol. 42, pp. 436–449, Nov. 2008.

[12] F. E. Wolfhagen, *The Train Unit Shunting Problem with Reallocation*. PhD thesis, Erasmus University Rotterdam, Mar. 2017.

[13] P. M. Jacobsen and D. Pisinger, "Train shunting at a workshop area," *Flexible Services and Manufacturing Journal*, vol. 23, pp. 156–180, June 2011.

[14] R. van den Broek, "Train shunting and service scheduling: an integrated local search approach," Master's thesis, Utrecht University, Aug. 2016.

[15] J. T. Haahr, R. M. Lusby, and J. C. Wagenaar, "Optimization methods for the train unit shunting problem," *European Journal of Operational Research*, vol. 262, pp. 981–995, Nov. 2017.

[16] E. Peer, V. Menkovski, Y. Zhang, and W. J. Lee, "Shunting trains with deep reinforcement learning," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 3063–3068, IEEE, Oct. 2018.

[17] D. van Cuilenborg, "Train unit shunting problem, a multi-agent pathfinding approach," Master's thesis, Technische Universiteit Delft, the Netherlands, 2020.

[18] D. Jekkers, *Train shunt planning using genetic algorithms*. PhD thesis, Erasmus University Rotterdam, Apr. 2009.

[19] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd ed., 2009.

[20] Z. Beheshti, "A review of population-based meta-heuristic algorithm," in *SOCO 2013*, 2013.

[21] S. Nesmachnow, "An overview of metaheuristics: Accurate and efficient methods for optimisation," *International Journal of Metaheuristics*, vol. 3, p. 320–347, Apr. 2014.

[22] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 28–39, Nov. 2006.

[23] H. G. Beyer and H. P. Schwefel, "Evolution strategies: A comprehensive introduction," *Natural Computing: An International Journal*, vol. 1, p. 3–52, May 2002.

[24] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[26] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers, 1997.

[27] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[28] H. H. Hoos and T. Stützle, "Stochastic local search algorithms: An overview," in *Springer Handbook of Computational Intelligence*, pp. 1085–1105, Springer Berlin Heidelberg, 2015.

[29] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann Publishers Inc., 2004.

[30] H. R. Lourenco, O. C. Martin, and T. Stutzle, "Iterated local search," *Handbook of Metaheuristics*, Feb. 2001.

[31] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[32] V. Kachitvichyanukul, "Comparison of three evolutionary algorithms: GA, PSO, and DE," *Industrial Engineering and Management Systems*, vol. 11, pp. 215–223, Sept. 2012.

[33] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, pp. 111–128, Sept. 2011.

[34] D. Ashlock, *Evolutionary Computation for Modeling and Optimization*. New York: Springer-Verlag, 2006.

[35] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*. Natural Computing Series, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[36] D. B. Fogel, "The advantages of evolutionary computation," 1997.

[37] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pp. 261–265, IEEE, Dec. 2016.

[38] M. Pirlot, "General local search methods," *European Journal of Operational Research*, vol. 92, pp. 493–511, Aug. 1996.

[39] R. Wehrens and L. M. Buydens, "Evolutionary optimisation: a tutorial," *TrAC Trends in Analytical Chemistry*, vol. 17, pp. 193–203, Apr. 1998.

[40] B. Doerr, E. Happ, and C. Klein, "Crossover can provably be useful in evolutionary computation," in *Proceedings of the Genetic and Evolutionary Computation Conference*, July 2008.

[41] T. Weise, Y. Wu, R. Chiong, K. Tang, and J. Lässig, "Global versus local search: the impact of population sizes on evolutionary algorithm performance," *Journal of Global Optimization*, vol. 66, pp. 511–534, Nov. 2016.

[42] K. Helsgaun, "General k-opt submoves for the Lin–Kernighan TSP heuristic," *Mathematical Programming Computation*, vol. 1, pp. 119–163, Oct. 2009.

[43] J. Schaller and J. M. Valente, "A comparison of metaheuristic procedures to schedule jobs in a permutation flow shop to minimise total earliness and tardiness," *International Journal of Production Research*, vol. 51, pp. 772–779, Feb. 2013.

[44] B. Possel, L. J. J. Wismans, E. C. Van Berkum, and M. C. J. Bliemer, "The multi-objective network design problem using minimizing externalities as objectives: comparison of a genetic algorithm and simulated annealing framework," *Transportation*, vol. 45, pp. 545–572, Mar. 2018.

[45] A. Pailla, A. R. Trindade, V. Parada, and L. S. Ochi, "A numerical comparison between simulated annealing and evolutionary approaches to the cell formation problem," *Expert Systems with Applications*, vol. 37, pp. 5476–5483, July 2010.

[46] M. A. Arostegui, S. N. Kadipasaoglu, and B. M. Khumawala, "An empirical comparison of Tabu Search, Simulated Annealing, and Genetic Algorithms for facilities location problems," *International Journal of Production Economics*, vol. 103, pp. 742–754, Oct. 2006.

[47] H. Youssef, S. M. Sait, and H. Adiche, "Evolutionary algorithms, simulated annealing and tabu search: a comparative study," *Engineering Applications of Artificial Intelligence*, p. 15, 2001.

[48] D. Nam and C. H. Park, "Multiobjective simulated annealing: A comparative study to evolutionary algorithms," *International Journal of Fuzzy Systems*, vol. 2, Jan. 2000.

[49] D. R. Westhead, D. E. Clark, and C. W. Murray, "A comparison of heuristic search algorithms for molecular docking," *Journal of Computer-Aided Molecular Design*, vol. 11, pp. 209–228, 1997.

[50] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Evolutionary Computation*, vol. 1, p. 67–82, Apr. 1997.

[51] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, Feb. 2012.

[52] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Computing Surveys*, vol. 45, pp. 1–33, June 2013.

[53] R. K. Ursem, "Diversity-guided evolutionary algorithms," in *Parallel Problem Solving from Nature*, pp. 462–471, Springer Berlin Heidelberg, 2002.

[54] D. Thierens and P. A. Bosman, "Optimal mixing evolutionary algorithms," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, p. 617–624, Association for Computing Machinery, 2011.

[55] G. Aalvanger, H. Luong, P. Bosman, and D. Thierens, "Heuristics in permutation gomea for solving the permutation flowshop scheduling problem," in *Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence*, pp. 146–157, Sept. 2018.

[56] M. Virgolin, T. Alderliesten, A. Bel, C. Witteveen, and P. A. N. Bosman, "Symbolic regression and feature construction with gp-gomea applied to radiotherapy dose reconstruction of childhood cancer survivors," in *Proceedings of the Genetic and Evolutionary*

Computation Conference, GECCO '18, p. 1395–1402, Association for Computing Machinery, 2018.

[57] M. C. van der Meer, B. R. Pieters, Y. Niatsetski, T. Alderliesten, A. Bel, and P. A. N. Bosman, "Better and faster catheter position optimization in hdr brachytherapy for prostate cancer using multi-objective real-valued gomea," in *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '18, p. 1387–1394, Association for Computing Machinery, 2018.

[58] B. Craenen, A. Eiben, and E. Marchiori, "How to Handle Constraints with Evolutionary Algorithms," in *Practical Handbook of Genetic Algorithms*, pp. 341–361, 2001.

[59] A. Eiben, "Evolutionary algorithms and constraint satisfaction: Definitions, survey, methodology, and research directions," in *Theoretical Aspects of Evolutionary Computing*, pp. 13–30, Springer Berlin Heidelberg, 2001.

[60] C. A. Coello Coello, "Constraint-handling techniques used with evolutionary algorithms," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO '16 Companion*, pp. 563–587, ACM Press, 2016.

[61] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. SSC-4(2), pp. 100–107, 1968.

[62] J. E. Hopcroft and R. M. Karp, "An n 5/2 algorithm for maximum matchings in bipartite graphs," 1973.

[63] C. E. Brodley, "Addressing the selective superiority problem: Automatic algorithm/-model class selection," 1993.

[64] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *J. ACM*, vol. 23, p. 665–679, Oct. 1976.

Appendix A

Glossary

List of Acronyms

CBC	Conflict-Based Crossover
EA	Evolutionary Algorithm
FOS	Family Of Subsets
GOMEA	Gene-pool Optimal Mixing Evolutionary Algorithm
HIP	Hybrid Integrated Planner
STAG	Shunt Train Activity Graph
TUSS	Train Unit Shunting and Servicing
TUSP	Train Unit Shunting Problem
VNS	Variable Neighbourhood Search

Appendix B

Ordering of movements

This Appendix discusses an issue that can arise with the ordering of movements when performing crossover with two solutions that have a different crossover. Applying crossover to the solutions in Figures B-1 and B-2, where you take unit 1 from Solution A and unit 2 from Solution B, while keeping the matching of solution 1 (indicated by the departure times printed underneath the departure nodes), yields the solution shown in Figure B-3.

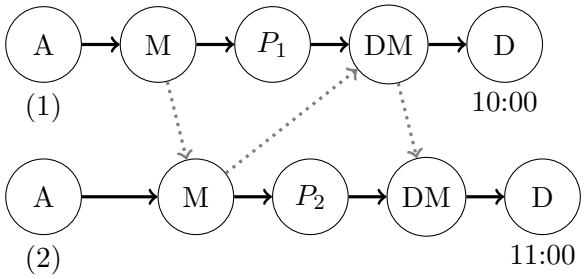

Figure B-1: Solution A

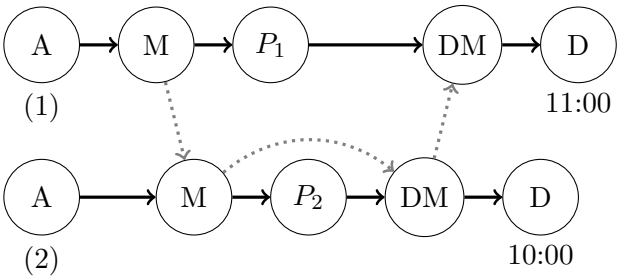

Figure B-2: Solution B

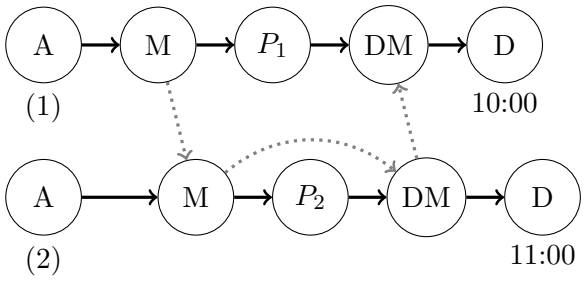

Figure B-3: New solution

In the new solution the departure movement of unit 2 is before the departure movement of unit 1, even though unit 1 leaves earlier. This happens because the movements are ordered based on their times in the original solution, and the departure movement of unit 2 in its original solution (which was solution 2) is 10:00 since there it was matched to the train leaving at that time. But in solution 1 unit 1 is leaving at 10:00, so when ordering the movements in the new solution there are two departure movements with the same departure time, while one of them is actually leaving later.

To address this problem, the times of the departure movements are kept intact from the solution from which the matching is used. For any unit that is then inserted from another solution during crossover (in this case, unit 2 from solution 2), for each movement it is checked if it is not after its departure movement. If this is not the case, the movement keeps its original time.

Appendix C

Additional results

Hardware	Lenovo P1	**Location**	Grote Binckhorst
Methods	HIP-8, CBC-8	**Problem instances**	50
Evaluation limit	1,600,000	**Train count**	41, 45
Time limit (min)	None	**Service tasks**	Yes

Table C-1: Setup of additional experiments.

Method	Pop. size	Solved (%)	Avg. time (sec)	
			Solved	*Unsolved*
CBC	8	55	909	1892
HIP	8	100	257	-

Table C-2: Results for Grote Binckhorst with 41 trains.

Method	Pop. size	Solved (%)	Avg. time (sec)	
			Solved	*Unsolved*
CBC	8	27	940	1782
HIP	8	100	302	-

Table C-3: Results for Grote Binckhorst with 45 trains.

Appendix D

Splits and combines

The crossover operator discussed in Section 4-4-1 does not address splits and combines as it currently stands. This appendix chapter shows how it can do so.

The issue with splits and combines lies therein that two solutions with different matchings can be mixed. Because they have different matchings, they will have different splits and combines. This needs to be addressed without making the solution invalid. This is done as follows:

- The matching is selected in its entirety from one of the two parent solutions (as is already the case in the current crossover).
- Each incoming train is split into its individual units on arrival by default. When a feasible solution has been found, this can be optimised to remove unnecessary splits.
- Every combine happens right before departure, and not sooner. Similarly to the previous point, this can be optimised when a feasible solution is found.
- If the matching is chosen from solution A, then for every train unit that is not selected from A, the last track that unit parks on in solution A is added to that unit right before its departure movement. This ensures that units that need to be combined before departure, are on the same track.

Figures D-1, D-2, and D-3 show an example. The former two show two parent solutions that are mixed together, while the latter shows the result of crossover of these two solutions, in the case where units 1 and 3 are taken from B and unit 2 is taken from A, and the matching is also taken from A. Because of this, the last parking movements of each train unit in the new solution is the same as in A, and as such trains that need to be combined are on the same track, as they were in the original solution.

In this case, this does introduce some inefficiencies, because train unit 1 moves from P_1 to P_5 and back to P_1 in the new solution, where it would be more efficient to just stay on track 1. These efficiencies can possibly be removed after a new valid solution has been created. This can be done by hand, by a dedicated subroutine, or by the local search (although the latter would not guarantee the removal of such inefficiencies).

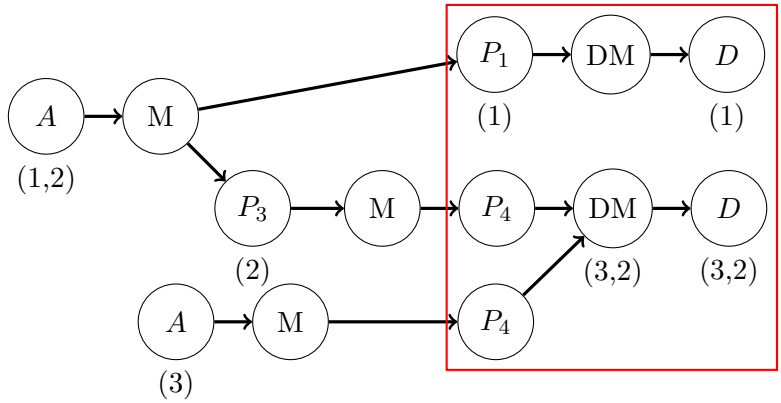

Figure D-1: Shunt Train Activity Graph (STAG) of Parent A. For simplicity, the order of the movements is not depicted. The nodes within the red box are all added to the child solution, regardless of which parent the unit is selected from.

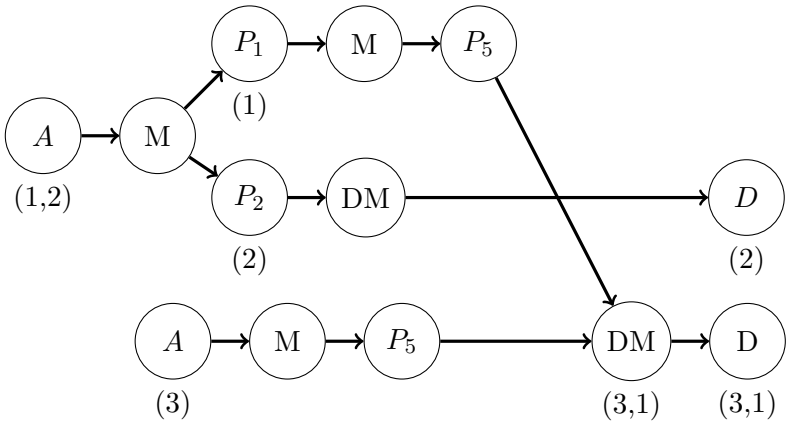

Figure D-2: STAG of Parent B. As can be noted from the different combines at the end, this solution has a different matching (indicated by the numbers denoting the train unit ids underneath the departure nodes).

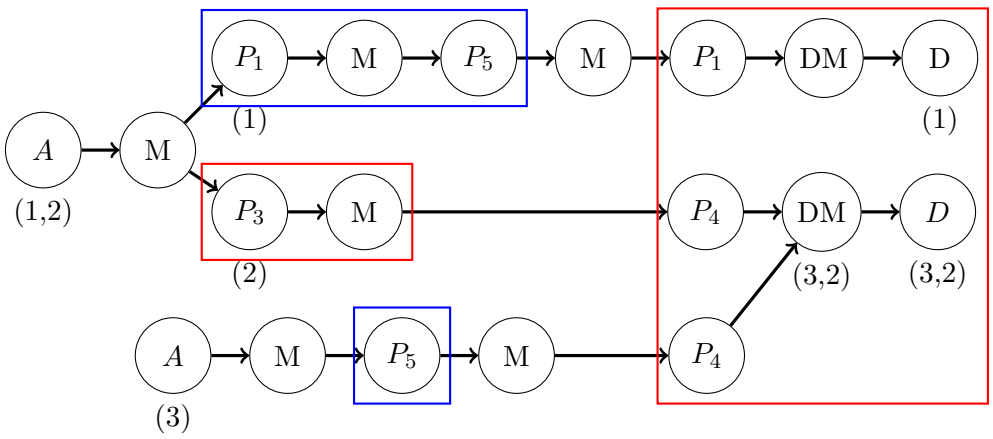

Figure D-3: STAG of new solution created by mixing Solution A and B: the matching is taken from A, unit 2 is selected from A, and units 1 and 3 are selected from B. The nodes in red boxes are taken from A, the nodes in blue boxes from B.

Appendix E

CBC for the open shop scheduling problem

This appendix offers a concept version of how the crossover operator designed for Train Unit Shunting and Servicing (TUSS) can be applied to the Open Shop Scheduling Problem (OSSP). In OSSP there are m machines and n jobs. Each job has a set of tasks. Each task i takes a predetermined amount of time t_i and has to be performed on a predetermined machine m_i. Each machine can only perform one task at a time, and the different tasks of a single job cannot be performed simultaneously. The problem is finding an assignment of tasks to machines such that the makespan is minimised. Table E-1 shows an example problem instance of OSSP.

Figure E-1 and Figure E-2 shows two solutions to this problem instance that are not optimal, and Figure E-3 shows a solution that is obtained by combining the aforementioned two solutions. The final solution is optimal. This shows a simple example how two solutions to OSSP can be combined to form a new solution, which is in essence a crossover.

The figures that show the example solutions can be represented by a graph that is very similar to a Shunt Train Activity Graph (STAG). Figures E-4, E-5, and E-6 show the graphs

Job	Task 1		Task 2	
	Machine	Time	Machine	Time
A	M_1	5	M_2	5
B	M_1	5	M_2	5
C	M_1	10	M_2	10

Table E-1: Example problem instance of OSSP. There are 2 machines on which tasks need to be performed, and there are 3 jobs, that each have 2 tasks. Jobs A and B each have two tasks that both take 5 time units, one of which needs to be performed on machine M_1 and the other on M_2. Job C has two tasks that both take 10 time units to complete, and also need to be performed on machines M_1 and M_2, respectively.

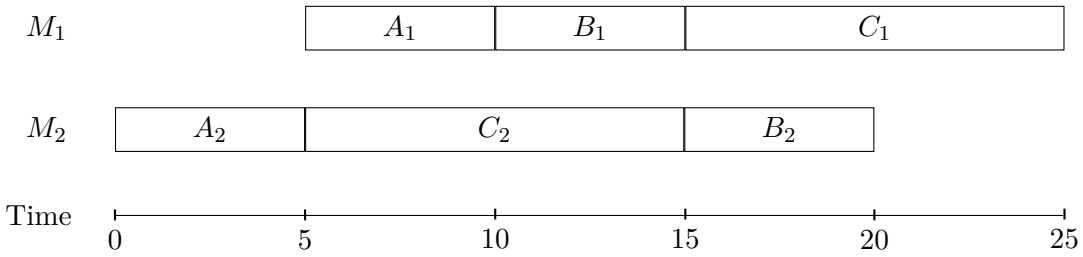

Figure E-1: An example configuration for the problem instance in Table E-1 with a makespan of 25. Note that the letter refers to the job and the subscript refers to a task from that job, e.g. A_2 refers to task 2 of job A. Job A_1 can only start on machine M_1 after A_2 on M_2 is finished, because tasks of the same job cannot be performed simultaneously.

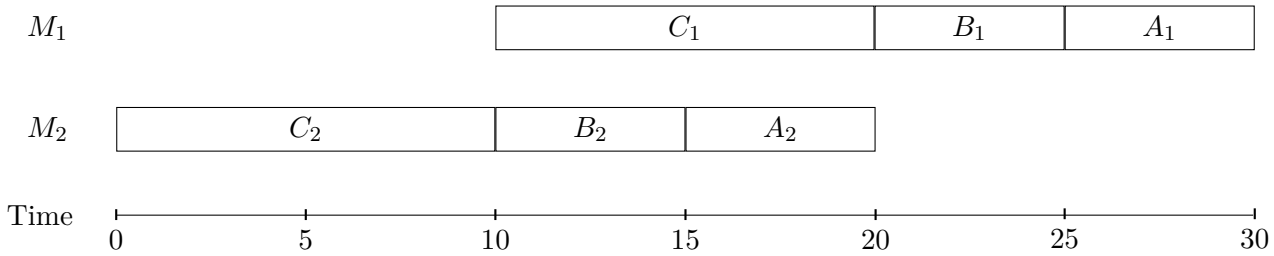

Figure E-2: A different configuration of tasks to machine for the problem instance in Table E-1, now with a makespan of 30. Similarly to the previous configuration, C_1 can only start after C_2 has been completed.

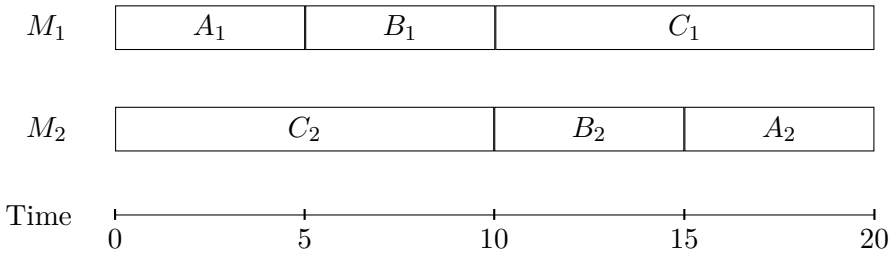

Figure E-3: A final configuration for the problem instance in Table E-1, now with a makespan of 20 (the lowest of the three configurations, and an optimal solution since none of the machines has any waiting time). This solution is obtained by taking the schedule of M_1 from the configuration in Figure E-1 and the schedule of M_2 from the configuration in Figure E-2.

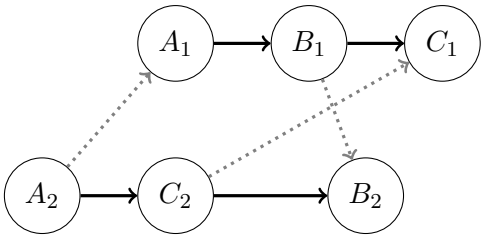

Figure E-4: This is a graph representation corresponding to the solution in Figure E-1. The black arcs indicate the precedence relationship between tasks on the same machine, while the dotted grey arcs indicate the precedence relationship between tasks of the same job.

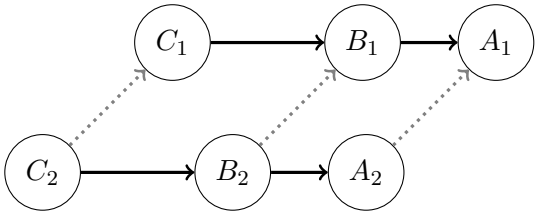

Figure E-5: This is a graph representation corresponding to the solution in Figure E-2.

corresponding to the configurations in Figures E-1, E-2, and E-3, respectively. Crossover can be applied to these graphs in the same manner as is done for TUSS. In this case, the schedule for a train unit in TUSS corresponds to the schedule of a machine here. Crossover is then performed by taking the schedules of some machines from one solution, and the schedules of other machines from the other solution.

To determine which machine schedule to select from which solution an option would be to look at the finishing time of both machines, and select the one with the earliest finishing time. Finishing time then corresponds to conflict cost of a train unit in TUSS (which is used in the heuristic of conflict-based crossover for TUSS). If the finishing time is equal, the solution from which the machine schedule is selected can be chosen at random; it is probably also possible to devise some additional or different heuristics for this step.

Diversity-based parent selection can also be applied here. Per machine, the number of swaps that are needed to get from one machine configuration to another can be calculated. If you do this for all machines between two solutions and sum this up, you get a diversity measure which is similar to the one used in Conflict-Based Crossover (CBC).

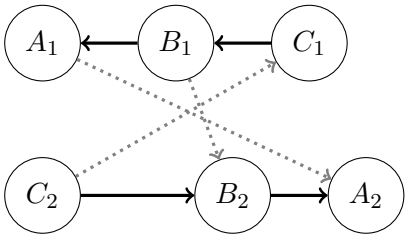

Figure E-6: This is a graph representation corresponding to the solution in Figure E-3. This is obtained by selected the schedule of M_1 from the first solution (the top 'row') and the schedule of M_2 from the second solution (the bottom 'row').