Comparing the performance of state-of-the-art algorithms for the nurse rostering problem

by

J.C.A. Faasse

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday November 15, 2024 at 12:30 PM.

Student number:	4677838	
Project duration:	March 1, 2024 – November 15, 2024	
Thesis committee:	Dr. ir. J.T. van Essen,	TU Delft, supervisor
	Dr. G.F. Nane,	TU Delft
	Dr. L. Berghman,	ORTEC

Preface

This thesis signifies my conclusion of the Master's program of Applied Mathematics at TU Delft, with specialization in Discrete Mathematics and Optimization. I wish to express my gratitude to those people who have made it possible for me to carry out the research presented here. First, to those at ORTEC: especially to my supervisor, dr. Lotte Berghman, for guidance and sharp questions, and to my daily supervisor, Eva van Rooijen, for enthusiasm, readiness to help and the effort put in to make the experiments possible; also to the Core Crocodylians, for moral support and letting me be a part of the team. Special thanks also to my supervisor from TU Delft, dr. ir. Theresia van Essen, for support and valuable feedback on writing and the project as a whole. Finally to dr. Tina Nane, for the kind willingness to be part of my thesis committee.

Jonathan Faasse
Rotterdam, November 2024

Abstract

The future prospect of healthcare workers in the Netherlands is worrisome, due to stressful working conditions and large expected personnel shortages. High quality personnel rosters have been shown to be able to alleviate this problem. The problem of creating personnel rosters that are of high quality in terms of how much they satisfy constraints regarding labor rules, work demand and personnel preferences, is denoted by the nurse rostering problem (NRP). This study aims to find an algorithm to solve the NRP that is suitable for implementation in a general automatic shift scheduler.

Firstly, literature on the NRP is reviewed, from which we conclude that single-solution based meta-heuristics are most suitable for this purpose. A categorization is made of different algorithm components, that are varied among different methods, namely construction methods, neighborhood structures, overall frameworks and perturbation methods. Secondly, based on the conclusions from the literature review, two construction methods, i.e. Construction-per-shift and Construction-per-employee, and two overall frameworks, i.e. Simulated Annealing and Variable Neighborhood Search, are implemented. Experiments are performed on nine data instances from a Dutch hospital, for which the problem description, in terms of hard and soft constraints, is drawn up to reflect real-world target cases. Different variations within the implemented methods are tested, from which general conclusions are drawn, mostly on the use of neighborhood structures within the overall frameworks.

Overall, Construction-per-shift greatly outperforms Construction-per-employee, Simulated Annealing slightly outperforms Variable Neighborhood Search, and the performance of the overall frameworks is largely independent of the preceding construction method. Based on the results, we conclude that both Simulated Annealing and Variable Neighborhood Search are stable and general methods, that can produce high quality rosters within a short time, making them suitable for implementation in a general automatic shift scheduler.

Potential future improvements could be found in additional algorithm adjustments, such as adaptive neighborhood probabilities for Simulated Annealing, targeted perturbation for Variable Neighborhood Search, or hard constraint relaxations.

Contents

1 Introduction

1.1. Societal relevance

The future prospect of healthcare workers in the Netherlands is worrisome. Studies by Statistics Netherlands (Centraal Bureau voor de Statistiek) have shown that healthcare workers have higher sick leave percentages than averaged over all sectors (7.3% compared to 5.3% in 2023, even more so in nursing specifically with 8.9% (CBS, 2024)), their job satisfaction is significantly impaired by the amount of stressful work and the lack of influence on their working hours, and half of them considered the workload too high in 2022 (CBS, 2022). Furthermore, a shortage of 195,000 healthcare employees is expected for 2033 (Helder, 2023). Together, these different factors can easily be imagined to form a downward spiral in overall healthcare working conditions.

Results reported by Rosenström et al. (2021), and listed by Xu and Hall (2021) suggest that more balanced and regular personnel shift schedules can reduce the expected employee sickness absence and fatigue levels. These findings motivate the development of methods that are able to generate high-quality personnel rosters, thereby improving the working conditions of healthcare workers.

1.2. Problem description

The problem of creating personnel rosters that are of high quality in terms of how much they satisfy constraints regarding labor rules, work demand and personnel preferences, is denoted by the nurse rostering problem (NRP). A roster for a given planning period consists of assignments for each nurse and each day in the planning period, whether that nurse has to work on that day or has a day off. On an assigned working day, a shift type and a skill type have to be specified. A shift type is defined by a start and end time. A skill type refers to a specific skill, qualification or position, that each employee does or does not have, e.g. the position of head nurse. The work demand is given by the coverage requirements, that specify how many nurses are required for each shift and skill type on each day of the planning period. The nurses can have both individual and collective contractual agreements, that specify, for example, minima and maxima of working hours, numbers of weekends worked or consecutive shifts or days off. Also, the shift types a nurse can be assigned to can be part of an individual contract. In the ideal case, one could always create rosters that satisfy all the given constraints. However, in practice this is generally not possible. Therefore, the constraints are typically divided in two categories, i.e. hard constraints and soft constraints. The hard constraints are those constraints that have to be satisfied in order for a roster to be acceptable by the user. The soft constraints are then those constraints that may be violated when necessary, as long as the hard constraints are satisfied.

Mathematically, the NRP can be formulated as a combinatorial optimization problem of assigning nurses to working days, shift types and skill types. The feasible region is defined as the set of solutions that satisfy all hard constraints. The objective function, in the context of the NRP also called the roster penalty, is given by the weighted sum of soft constraint violation penalties, where the weights reflect the relative importance of the soft constraints.

1.3. Complexity NRP

Den Hartog et al. (2023) showed that an NRP with 4 shift types (including day-off), coverage constraints, day-off requests and forbidden shift sequences of length 2 is strongly $\mathcal{NP}$-hard. Here, a forbidden shift sequence of length k is a combination of k shift types that may not be assigned on k successive days. For example, a forbidden shift sequence of length 2 could be that a night shift may not be followed by a day shift on the next day. Since most practical cases have many more shift types and constraints, we can assume that real-world nurse rostering problems are strongly $\mathcal{NP}$-hard.

1.4. ORTEC

This study was done in cooperation with, and under supervision of ORTEC. ORTEC is a company aimed at making a positive societal impact using mathematics, by offering technical consulting services, developing commercially available optimization software, and by conducting research on the underlying algorithms. This is done in a variety of sectors, among which transportation, finance, energy and healthcare.

1.5. Typical target cases

Because the performance of an algorithm is highly dependent on its application, it is relevant to state the characteristics of a typical target case. These characteristics are provided by ORTEC, based on their experience with customers of their scheduling software, and are shown in Table 1.1. The most important hard constraints that are currently considered are given by labor rules from the Dutch Working Hours Act (Arbeidstijdenwet, 2022, §5.2).

Table 1.1: Characteristics of typical target cases.

Characteristic	**Typical target case**
Number of employees	50-150
Number of shift types	3-10
Number of skill types	5-20
Planning period	1 month
Hard constraints	Labor rules
	Contractual working hours
	Required skill levels met
Soft constraints	Coverage requirements
	Requested days/shifts on/off
	Preferred shift and day-off sequence lengths

1.6. Research questions

Whereas some literature can be found of researchers applying a shift scheduling algorithm to a real-life hospital case, it is not feasible to supply a wide spread of hospitals with shift scheduling software in this way. In order to supply a wide variety of potential users, more general and widely available scheduling software is required. The aim of this research is to find out which algorithm would be most effective in a general, widely applicable automatic shift scheduler. Such an algorithm should have at least the following characteristics:

- **stability:** results from different runs on the same instances should not drastically vary, in order to guarantee its reliability;
- **generality:** the algorithm is able to handle a wide variety of problem specifics and input data, such as the objective function, which (hard and soft) constraints are required, the number of shift types and the number of employees;
- **give high quality rosters:** most of the scheduling is still done manually, partly because schedulers often think that they 'know what is going on' and an algorithm cannot incorporate that. Previous work by Van Rooijen (2023) has shown that nurse preferences can be incorporated in the NRP. In order for schedulers to become convinced of the relevance of an automatic scheduler, it should be able to generate high quality rosters, while incorporating these employee preferences;
- **short computation times:** the algorithm is able to generate rosters within a reasonable amount of time. Note that, in ORTEC's experience, users can have different views on what is a reasonable amount of time,

based on how the algorithm is used. Some users run the algorithm once per scheduling period, and therefore, accept long run times of multiple hours. Others run the algorithm multiple times, to see the effect of changes in the input, and thus, require a short run time. To account for the latter user group, the algorithm should be able to generate rosters in the timescale of minutes.

In this context, we formulate the following research question:

What algorithm is suitable for implementation in a commercial automatic shift scheduler?

To answer this question, we first require answers to the following subquestions:

- *What is the state-of-the-art in algorithms to solve the NRP?*
- *How do different combinations of promising algorithm components compare in terms of solution quality, when applied to real-world target cases?*

2 Literature review

2.1. Literature search approach

First, relevant literature reviews on the NRP were sought for, to get an overview of the field. The search engines Scopus and Google Scholar were used. Next, relevant research articles were found in the relevant reviews and by considering articles referenced by those articles or by which they are cited. Finally, Scopus and Google Scholar were searched with combinations of search terms including - but not limited to - “nurse rostering”, “shift scheduling”, “algorithm”, “approach”, “solution method”. Mostly articles published no earlier than 2008 were considered. Further filtering was done based on the obtained results and how much the applications resembled the typical target applications.

2.2. Benchmarks

Many algorithms found in literature are tested on sets of benchmark instances. Three benchmarks are found to be relevant, based on how much they resemble the typical target cases and how much they are used in literature. Two of them come from the International Nurse Rostering Competitions: INRC-I and INRC-II. The third is the Shift Scheduling benchmark set. The benchmark sets are described below, and their characteristics are summarized in Tables 2.1 and 2.2.

2.2.1. INRC-I

In 2010, the first International Nurse Rostering Competition (INRC-I) was written out by Haspeslagh et al. (2010). The problem to be solved by the competitors was to create rosters that satisfy the hard constraints of single-shift-per-day and required shift coverage, while minimizing the penalties of soft constraints violations. The competition consisted of solving sixty instances, divided over three tracks: sprint, medium and long. Time limits of 10 seconds, 10 minutes and 10 hours were set on the respective tracks. See Tables 2.1 and 2.2 for further characteristics and constraints, and the report by Haspeslagh et al. (2014) for the complete formulation, competition regulations and results. Also after the competition was concluded, the INRC-I benchmark was often used by authors to test the performance of their algorithms. Note that this benchmark has coverage requirements as hard constraints, instead of labor rules, which are typical in the target cases. The instances are available at the INRC-I website[1].

2.2.2. INRC-II

The second International Nurse Rostering Competition (INRC-II) was held in 2014-2015, written out by Ceschia et al. (2015). Competitors were challenged to devise algorithms to solve a multi-stage NRP. The hard constraints were given by a minimum shift coverage including required skill types, forbidden shift type successions and single-shift-per-day. The objective was to minimize the total penalty for violations of soft constraints. In the multi-stage setting, the instances were solved one week at a time, without knowledge on the successive weeks. In total, 88 instances were used in the competition, and the time limit per week in the planning horizon was set between 5 and 50 minutes, specified by the number of employees in the instance. Tables 2.1 and 2.2 show further characteristics of the INRC-II benchmark, and we refer to the report by Ceschia et al. (2019) for the full description of the competition procedures and results. After the competition, the INRC-II benchmark was used

[1]https://nrpcompetition.kuleuven-kulak.be/instances-results/

both in its multi-stage formulation, as well as in a static version, which has the same problem formulation, but leaves out the multi-stage setting by considering the whole planning horizon at once. Note that, similar to INRC-I, this benchmark has mostly coverage requirements as hard constraints, instead of labor rules, which are typical in the target cases. The instances are available at the INRC-II website[2].

2.2.3. Shift Scheduling

Curtois and Qu (2014) introduced the Shift Scheduling benchmark instances, which were designed for their resemblance to real-world cases and ease of use. The underlying problem is to create rosters that minimize penalties resulting from coverage shortages and surpluses, and ungranted day-on/off requests. The set of hard constraints consists of single-shift-per-day, fixed days off, forbidden shift type successions and minima and maxima of working hours, weekends and consecutive shifts and days off. Note that all hard constraints can be checked for each employee separately. The instances vary greatly in size. The smallest instance has a planning period of two weeks, with 8 employees and 1 shift type, whereas the planning periods in the largest instances span a full year, with up to 150 employees and 32 shift types. Note that skill types are included in the shift types in this benchmark. Further characteristics of the Shift Scheduling benchmark are shown in Tables 2.1, 2.2 and A.6. The Shift Scheduling benchmark is frequently used in recent studies to test and compare algorithm performances. Note that although the type of hard constraints in this benchmark is similar to those in the typical target cases, the number of soft constraints is relatively low. The instances are available at the shift scheduling benchmarks website[3].

Table 2.1: Hard and soft constraints of the problem formulations used in the INRC-I, INRC-II and Shift Scheduling benchmarks.

INRC-I	**INRC-II**	**Shift Scheduling**
Hard constraints	***Hard constraints***	***Hard constraints***
Single shift per day	Single shift per day	Single shift per day
Required shift coverage	Minimal shift coverage	Forbidden shift type successions
	Required skill levels met	Max # of shifts
Soft constraints	Forbidden shift type successions	Min/max # of working hours
Required skill levels met		Min/max # of consecutive working days
Min/max # of shifts	***Soft constraints***	Min/max # of consecutive days off
Min/max # of consecutive working days	Desired shift coverage	Fixed days off
Min/max # of consecutive days off	Min/max # of shifts	
Max # of working weekends	Min/max # of consecutive working days	***Soft constraints***
Max # of consecutive working weekends	Min/max # of consecutive days off	Coverage shortage
Complete weekends	Max # of working weekends	Coverage surplus
Identical weekend shifts	Complete weekends	Day-on/off requests
Min # of days off after night shift	Day-on/off requests	
Day-on/off requests	Shift-on/off requests	
Shift-on/off requests		
Alternative skills		
Unwanted shift patterns		

Table 2.2: Characteristics of the INRC-I, INRC-II and Shift Scheduling benchmark instances.

Characteristic	**INRC-I**	**INRC-II**	**Shift Scheduling**
Number of employees	10-50	30-120	8-150
Number of shift types	3-5	4	1-32 (incl. skill types)
Number of skill types	1-2	4	-
Planning horizon	4 weeks	4-8 weeks	2-52 weeks
Setting	Single-stage	Multi-stage	Single-stage

[2]https://mobiz.vives.be/inrc2/?page_id=20
[3]7http://www.schedulingbenchmarks.org/nrp/

2.3. Algorithms

The NRP is a widely studied problem, and many different approaches for obtaining high-quality rosters have been developed. Comprehensive reviews are given by Burke et al. (2004) and, more recently, by Ngoo et al. (2022). Other recent reviews that (partly) cover the NRP are drawn up by, Özder et al. (2020), Lan et al. (2021) and Xu and Hall (2021). Here, we summarize the main contributions found in literature, divided over the categories of meta-heuristics, hyper-heuristics and mathematical optimization.

2.3.1. Meta-heuristics

Variable Neighborhood Search Variable Neighborhood Search (VNS) is a meta-heuristic framework, where solutions are iteratively improved by applying moves sampled from various neighborhood structures. In the context of the NRP, an example of a neighborhood structure can be all moves where a single shift assignment of one nurse is swapped with the shift assignment of another nurse on the same day. In most cases, a perturbation mechanism is activated after a number of iterations without improvement, in order to escape from local optima. Many different variants of VNS have been applied to the NRP. Lü and Hao (2012) developed an adaptive VNS, which focused more on diversification as opposed to intensification, if there have been few improvements in recent iterations, and vice versa. Tassopoulos et al. (2015) implemented a two-phase approach. First, the working days and days off were determined for all employees, followed by the assignment of specific shift types to the fixed working days. Zheng et al. (2017) developed a randomized VNS, where in each iteration, first one of two neighborhoods is chosen randomly, followed by a move in the chosen neighborhood. Meignan and Knust (2019) introduced a neutrality-based Iterative Local Search (ILS) variant. The authors state that in the NRP, local optima are often part of plateaus of neighboring solutions of similar quality. Thus, a plateau exploration is proposed after the local search, in order to escape local optima without significantly degrading the solution. When applied to the INRC-I benchmark, Lü and Hao (2012), Tassopoulos et al. (2015), Zheng et al. (2017) and Meignan and Knust (2019) obtain comparable, and competitive results. Notably, the randomized VNS produces the most stable results among these approaches, in terms of standard deviation after multiple runs.

Abdelghany et al. (2021a) combined a VNS with a dynamic programming method within a ruin-and-recreate perturbation framework. In another approach, Abdelghany et al. (2021b) proposed a two-stage VNS approach, using the same neighborhoods. The first stage aims to minimize the coverage shortages and surpluses, using coverage-focused neighborhoods. In the second stage, the solution is further improved with respect to all soft constraint violations. Applied to the Shift Scheduling benchmark, both approaches produce competitive results, with the single-stage method outperforming the two-stage version. Goh et al. (2022) developed an ILS method, in which a Monte Carlo Tree Search is implemented to quickly find an initial feasible solution, which is then improved in the ILS. On the Shift Scheduling benchmark, strong results are obtained.

Deviating from standard VNS approaches, Burke et al. (2013) proposed a Variable Depth Search (VDS) variant. In the VDS, multiple neighborhood swaps are chained together, in order to find compound moves that collectively improve the solution. Applied to the INRC-I and Shift Scheduling benchmarks, reasonably good results are obtained.

Simulated Annealing Ceschia et al. (2020) developed a two-stage Simulated Annealing approach to solve the NRP. In the first stage, moves are applied until a feasible solution is found, allowing moves to infeasible solutions. The second stage aims to improve the solution with respect to the soft constraint penalties, while remaining feasible. Their approach is applied to the static INRC-II benchmark, and achieves competitive results. A single-stage variant of the algorithm was later applied by Ceschia et al. (2023) to 34 real-world instances from Italian hospitals, where it was able to obtain good results.

Population-based local search Abuhamdah et al. (2021) introduced a population-based local search method. First, an initial population of solutions is initialized, each with a direction value, which tracks the amount of improvement and degradation of the solution quality. In each iteration, a local search is applied to the solution with the highest direction value, i.e. the solution which has been improved the most. If a solution has not been improved over a specified number of local search iterations, it is replaced by the best solution found so far, and perturbed for diversification. A Multi-Neighborhood Particle Collision Algorithm combined with an Adaptive Randomized Descent Algorithm (MPCA-ARDA) is used as the local search technique. MPCA-ARDA can accept worsening solutions, but only up to a certain threshold, which is determined by an average of recently accepted solution qualities. Very competitive results are reported for the INRC-I benchmark, albeit under relaxed time limits.

Harmony Search Algorithms Harmony search algorithms are population-based methods that imitate the process of musical improvisation. First, a population of solutions is initialized, the so-called harmony memory. Then, new solutions are created in an iterative framework, built up from parts of the solutions in the harmony memory and random assignments. Such a newly created solution is then improved by pitch adjustment, i.e. a local search. Finally, the harmony memory is updated, by adding the newly created solution, and subsequently removing the worst solution. Hadwan et al. (2013) implemented a harmony search algorithm for the NRP, and obtained reasonably good results on a set of old benchmark cases. Awadallah et al. (2014, 2017) developed several variants of harmony search. In the best performing variant, the pitch adjustment was replaced by an intensive hill climbing local search method. Reasonably good results are obtained on the INRC-I benchmark instances.

Bee Colony Optimization Inspired by the foraging strategy employed by bee colonies, bee colony optimization algorithms are population-based methods where a parallelized local search is combined with global information sharing. First, a population of solutions is initialized, the so-called food source memory. Then, in an iterative process, first all solutions in the food source memory are improved by the employer bees, i.e. a local search for each solution. Among the improved solutions by the employer bees, the onlooker bees probabilistically choose a solution from the food source memory, with higher probability for high quality solutions, to which an extra round of improvements is applied. Awadallah et al. (2015) implemented bee colony optimization, where the employer bees were replaced by a hill climbing local search. Applied to the INRC-I benchmark, reasonably good results are obtained.

Ant Colony Optimization Ant Colony Optimization algorithms follow the behavior that ants exhibit in their collective search for food. The ants perform individual explorations in search for food, and report their findings by secreting pheromones on the way back to the colony: the better the quality of the food source, the more pheromones are secreted. Subsequent ants are more likely to explore paths with more pheromones. The pheromones evaporate over time, ensuring continued diversification of the search. Jaradat et al. (2019) proposed an ant colony optimization approach, where the exploration is guided not only by the pheromones, but also by a memory of elite solutions. Applied to the INRC-I benchmark, competitive results are obtained.

2.3.2. Hyper-heuristics

Hyper-heuristic algorithms are characterized by employing a procedure that iteratively selects low-level heuristics to be applied to the solution. Kheiri et al. (2021) implemented a hidden Markov model to select effective sequences of different swapping neighborhoods and ruin-and-recreate mechanisms. Their approach came in third place in the INRC-II competition, and ranked first among both the methods without an IP solver and those that produced a feasible solution in each run.

2.3.3. Mathematical optimization

In the category of mathematical optimization (MO), we consider all algorithms which are based mainly on integer programming (IP) techniques. This includes both methods that solely use IP, and so-called matheuristics, that combine an MO technique with a heuristic or meta-heuristic approach.

Several MO approaches have been shown to be able to generate high quality solutions to nurse rostering problems. Burke and Curtois (2014) developed a branch-and-price method that was able to obtain the optimal solution in many of the instances of the INRC-I. However, when applied to the Shift Scheduling benchmark, it finds good solutions for the smaller instances, but does not manage to find a feasible solution for most medium sized and large instances before it runs out of memory.

Valouxis et al. (2012) developed a two-phase approach, where first the employees are assigned to working days, and subsequently, with the working days fixed, to specific shift types. Their algorithm was the winner of the INRC-I. Both Römer and Mellouli (2016) and Legrain et al. (2020b) formulate the NRP as a (network-)flow model. Römer and Mellouli (2016) solved it directly with an IP solver, whereas Legrain et al. (2020b) applied a branch-and-price approach. Römer and Mellouli (2016) and Legrain et al. (2020b) obtained first and second place in the INRC-II, respectively. However, even though they received the best scores in the competition, both algorithms did not succeed in finding a feasible solution in each run.

Other approaches hybridize mathematical optimization techniques with other heuristic or meta-heuristic methods. Burke et al. (2010) combined IP with variable neighborhood search, such that a feasible solution is generated by an IP solver, which is then improved by a VNS. Rahimian et al. (2017) integrated the IP solver in a ruin-and-recreate framework within a VNS. Finally, the best solution obtained in the VNS is fed into the IP solver, in order to improve the overall solution. This allowed their approach to obtain very good results for most of the instances in the Shift Scheduling benchmark. Only for the largest instances, the results were poor. Chen et al. (2023) used an approach similar to Rahimian et al. (2017), including an IP-based ruin-and-recreate framework. However, instead of a traditional VNS, they employed a deep neural network to choose the neighborhood used in each iteration. Good results were obtained when applied to the Shift Scheduling benchmark. Turhan and Bilgen (2020) presented a combination of fix-and-relax, Simulated Annealing and fix-and-optimize. An initial solution is generated by decomposing the problem and solving the sub-problems with the fix-and-relax mechanism. Simulated Annealing iterations and fix-and-optimize were then applied in an alternating fashion to improve the solution. Both the fix-and-relax and the fix-and-optimize mechanisms used an IP solver to construct the solution. Applied to the small and medium sized instances of the Shift Scheduling benchmark, competitive results are obtained.

2.3.4. Comparison of experimental results

Experimental results on the INRC-I and Shift Scheduling benchmarks are shown in Appendix A. Note that for the many instances in the INRC-I benchmark, best known solutions are obtained more often than not by competitive methods. Therefore, performances are often compared based on the number of instances on which the best known solution is found. As can be seen in Table A.5, the population-based methods by Jaradat et al. (2019) and Abuhamdah et al. (2021) performed very well on the INRC-I benchmark. However, it is hard to estimate how much this was influenced by their time limits, which were unreported and relaxed, respectively. Furthermore, the VNS variants by Lü and Hao (2012), Tassopoulos et al. (2015), Zheng et al. (2017) and Abdelghany et al. (2021a) obtained good results. Compared to the other methods, the VDS by Burke et al. (2013), the IP-based approach by Valouxis et al. (2012) and the population-based methods by Awadallah et al. (2015) and Awadallah et al. (2017) were not competitive. Meignan and Knust (2019) are not included in the tables for the INRC-I benchmark, since they only reported their average results and standard deviations. They obtained a better average result in only one instance, compared to Zheng et al. (2017). Standard deviations were reported by Lü and Hao (2012), Tassopoulos et al. (2015), Awadallah et al. (2015), Awadallah et al. (2017), Zheng et al. (2017), Jaradat et al. (2019) and Meignan and Knust (2019), among which Zheng et al. (2017) generally obtained the lowest standard deviation.

Regarding the INRC-II benchmark in the multi-stage formulation, the results by the IP-based methods of Römer and Mellouli (2016) and Legrain et al. (2020b) led them to first and second place in the competition, respectively. However, they both did not manage to find a feasible solution in each run for every instance. Conversely, the hyper-heuristic approach by Kheiri et al. (2021), awarded with the third place in the competition, had competitive results compared to Römer and Mellouli (2016) and Legrain et al. (2020b), and found a feasible solution in each run. No methods were found that generally outperformed these methods in this setting. In the static version of this benchmark, both Legrain et al. (2020a) and Ceschia et al. (2020) obtained competitive results. The branch-and-price-based method by Legrain et al. (2020a) outperformed the Simulated Annealing approach by Ceschia et al. (2020) in most cases. However, Ceschia et al. (2020) gained an advantage in cases with a longer planning period and shorter computation time.

Table A.7 shows the results of several algorithms on the Shift Scheduling benchmark with a time limit of 60 minutes, which is the most commonly used time limit for this benchmark. The best known solutions were obtained from the shift scheduling benchmark website[4]. Results by the Gurobi IP solver, as reported by Burke and Curtois (2014) and Goh et al. (2022) were added for comparison. It is clear that the IP-based methods by Rahimian et al. (2017), Turhan and Bilgen (2020) and Chen et al. (2023) perform very well in this setting, of which the IP + VNS approach by Rahimian et al. (2017) in particular. Among the other methods, the VNS/ILS approaches by Abdelghany et al. (2021a) and Goh et al. (2022) obtain good results, and generally outperform those by Burke and Curtois (2014) and Abdelghany et al. (2021b).

Several methods were also tested on the Shift Scheduling benchmark using a time limit of 10 minutes, see Table A.8. We find that also in this case, the IP-based methods by Rahimian et al. (2017) and Turhan and Bilgen (2020) perform well. However, their advantage with respect to Goh et al. (2022) is smaller, and they are outperformed on the larger instances under this reduced time limit.

2.4. Conclusion & discussion

Among the meta-heuristics with competitive performance, the single-solution-based methods are most prevalent. Thus, it appears that their advantage in terms of exploitation often outweighs the benefit that population-based methods have in terms of exploration / diversification. In contrast, most population-based methods manage to obtain good results, but their performance in terms of exploitation seems to suffer too much from the computational

[4]http://www.schedulingbenchmarks.org/nrp/

costs of managing and improving a population of solutions to outperform the single-solution-based methods. However, the question remains whether the same would be observed in cases which are more highly constrained than the benchmark instances.

Overall, IP techniques have been shown to perform very well on small to medium sized instances. Furthermore, IP techniques that are hybridized with another (meta)heuristic framework perform better than 'simple' IP methods, because they can combine their strengths of diversification and convergence, respectively. However, IP techniques cannot always guarantee to find good solutions or solution improvements or even feasible solutions within a reasonable time, especially once the (sub-)problems become large. This generally causes poor performance of IP-based methods on large instances, and reduces general reliability in terms of finding a feasible solution. Combined with the fact that most IP techniques rely on third-party software, they are deemed unsuitable for implementation in a general automatic shift scheduler.

Drawing further conclusions on the performance of different algorithms is made difficult by several issues:

- Not all methods are applied to the same model. The formulation of the NRP, in terms of its hard and soft constraints can have a large impact on the effectiveness of the algorithms. Different benchmarks are used in literature, which all have different problem formulations, and most algorithms are tested on only one of them, or only on instances obtained from a specific application that is not considered by others.

- The experimental setups vary among different studies. Most algorithms are tested using specified time limits. However, differences in hardware and software implementations can have a large impact on the computation speed, and because of that on the overall performance within those time limits. Thus, when algorithms have differences in performance, it is hard to estimate how much of that difference can be ascribed to the algorithms themselves.

- The algorithms require a large number of design choices. Between any two algorithms found in literature, multiple differences in their methodologies can be identified, e.g. different construction mechanisms, neighborhood structures or move acceptance criteria. Thus, when algorithms have differences in performance, it is hard to estimate which part(s) of the methodology these differences can mostly be ascribed to.

The issues described above motivate a comparative study of different combinations of algorithm components, applied to a well-defined, representative model with corresponding data sets, on a fixed experimental setup.

3 Categorization of algorithm components

In order to conduct a structured comparative study on combinations of parts of different solution methodologies, we categorize the different components that can typically be identified. In each category, several different design choices are listed, limited to those that were deemed the most promising based on the results in literature. Ultimately, all well-performing algorithms are built around local search techniques, and the specific algorithm components used are tailored to the considered problem formulation.

3.1. Overall framework

In most studies, an overall framework of the algorithm can be identified that characterizes the method as a whole. Those used by well-performing algorithms are listed below. Note that all of them are ultimately built upon local search techniques.

- Variable Neighborhood Search (VNS): in VNS, solutions are iteratively improved by applying moves sampled from various neighborhood structures. Different variants of VNS can be applied (Abdelghany et al., 2021a, 2021b; Goh et al., 2022; Zheng et al., 2017).
- Hidden Markov model-based hyper-heuristic: in this hyper-heuristic framework, a random sequence of low-level heuristics is chosen in each iteration. The low-level heuristics can use both applications of normal neighborhood structures or perturbation mechanisms. The probabilities of the choice of low-level heuristics are updated after each iteration, based on whether the chosen sequence managed to improve the solution (Kheiri et al., 2021).
- Simulated Annealing (SA): SA is a well-known method that is widely applicable. It iteratively chooses random moves in the neighborhood of the current solution, and accepts each move with a probability that is determined by the effect it has on the solution quality. If the move improves the solution, it is always accepted. Otherwise, the probability of acceptance becomes smaller, the larger the negative effect on the solution (Ceschia et al., 2020, 2023).
- Population-based local search: in a population-based local search, a population of solutions is kept in memory. In each iteration, a local search is applied to one of the solutions. The effects of the local searches on the solution qualities, which can be positive or negative, are tracked cumulatively in a so-called direction value. In each iteration, the solution with the highest direction value is chosen. Additionally, solutions which have not improved over a specified number of iterations are replaced by a newly generated solution in the neighborhood of the best solution found thus far (Abuhamdah et al., 2021).

3.2. Construction of initial solution

By far the most algorithms found in literature apply some form of greedy heuristic to generate an initial solution (population), which is then fed into the main method. Additionally, the focus of these methods mostly lies on finding a feasible solution, rather than a solution which is already of high quality. Consequently, the choice of construction method is mostly determined by which hard constraints are present in the model to which the algorithms are applied. In the case of hard coverage requirements, usually all necessary shifts are assigned to employees one by one, either randomly (Ceschia et al., 2020, 2023; Kheiri et al., 2021; Lü and Hao, 2012; Meignan and Knust, 2019; Tassopoulos et al., 2015), greedily (Burke et al., 2013) or via a heuristic ordering of shifts and/or employees (Awadallah et al., 2015, 2017; Burke et al., 2008). In contrast, building an initial solution from feasible rosters for each employee individually is more common in the case of hard constraints concerning labor rules (Abdelghany et al., 2021a, 2021b; Goh et al., 2022; Rahimian et al., 2017). Rahimian et al. (2017) report that both a high-quality, and a completely randomized initial solution lead to worse overall performance than a mediocre one constructed by a simple greedy heuristic. Furthermore, Goh et al. (2022) accredit their algorithm's good performance partly due to the speed of their Monte Carlo Tree Search construction method, leaving more time for the rest of the algorithm to improve the solution. These observations motivate the use of a fast construction algorithm that can find initial solutions that are at least better than a completely randomized roster.

3.3. Neighborhood structures

Most methods require the use of neighborhood structures in their search for solution improvements. A neighborhood structure is characterized by a specific type of operation, e.g. swapping the shifts of two nurses on the same day. The neighborhood of a solution can then be defined as the set of solutions that can be reached by applying the specified operation exactly once, somewhere in the current solution. Several different neighborhood structures are used in well-performing methods, characterized by the operations listed below.

- Vertical swaps: the shift assignments of two nurses are swapped on the same day or block of days. A special case of this operation can be to swap the complete weekends of two nurses.
- Horizontal swaps: one or more shift assignments of a nurse are swapped with shift assignments that come earlier or later in the planning period.
- Change operations: change the shift assignment of one nurse into another shift assignment on one or more days.
- Coverage-focused operations: specifically assign currently under-covered shifts to nurses who are free on the corresponding days. As a variant of this, such an assignment can be combined with unassigning the same nurse on another working day, which could be necessary to comply with working hours constraints. Note that these operations are special cases of the change operation.

As was the case for the construction methods, the choice of neighborhood structures often depends on the considered hard constraints. In the case that only coverage requirements constitute the hard constraints, vertical operations will never cause a solution to turn infeasible, as the number of assigned shifts on the corresponding days does not change. Thus, most methods applied to such a scenario only use vertical operations (Abdelghany et al., 2021a; Burke et al., 2013; Lü and Hao, 2012; Meignan and Knust, 2019; Tassopoulos et al., 2015; Zheng et al., 2017), and thereby never have to check feasibility of the neighborhood moves. Horizontal and change operations are more common in cases with hard constraints concerning working hours instead (Abdelghany et al., 2021a, 2021b; Ceschia et al., 2023; Goh et al., 2022).

Move acceptance criteria Closely related to the choice of neighborhood structures are the move acceptance criteria, i.e. how the choice is made to accept or reject a move in a solution's neighborhood. A first design choice in this sense is whether a neighborhood structure is exhaustively searched, to find and accept the best possible move in the neighborhood, or to accept the first improving move that is encountered. Although an exhaustive search can result in larger improvements per accepted move, it does come with an additional computational cost, which can become extensive in the case of large neighborhoods. Note that this choice is not applicable to methods that already have inherent move acceptance criteria, such as Simulated Annealing.

A second design choice is whether to accept moves toward solutions that are infeasible with respect to the hard constraints. Obviously, an infeasible solution cannot be returned as the final output. However, theoretically, accepting intermediate infeasible solutions can possibly lead to better final feasible solutions that are otherwise hard or even impossible to reach when only allowing moves to other feasible solutions. On the other hand, in a highly constrained solution space, it might be hard to escape from an infeasible region. In practice, most methods maintain feasibility throughout the complete algorithm (Abdelghany et al., 2021a, 2021b; Awadallah et al., 2015, 2017; Burke et al., 2010, 2013; Goh et al., 2022; Jaradat et al., 2019; Kheiri et al., 2021; Lü and Hao, 2012; Meignan and Knust, 2019; Tassopoulos et al., 2015; Zheng et al., 2017). Ceschia et al. (2020) initialize their

Simulated Annealing approach with a solution that is not guaranteed to satisfy all hard constraints, and allow for moves to other infeasible solutions in the first stage of their algorithm. Once a feasible solution is found, it is improved while maintaining feasibility. In their later version, Ceschia et al. (2023) allow for moves to infeasible solutions throughout their whole method. In both cases, the 'degree of infeasibility' is incorporated into the objective function, in the form of hard constraint violation penalties, similar to those used for the soft constraints. Weights for the hard constraint violations were chosen to reflect their large relative importance.

3.4. Perturbation / improvement methods

One of the challenges in optimizing over highly constrained solution spaces is to not get stuck in local optima. Therefore, many algorithms contain some mechanism to perturb the current solution (population), often activated after a certain number of iterations without improvement. For some methods, perturbation simply consists of accepting several non-improving neighborhood moves, that would otherwise not be accepted (Abdelghany et al., 2021b). Others apply slight variations of the regular neighborhoods (Goh et al., 2022; Tassopoulos et al., 2015), or a special perturbation neighborhood (Zheng et al., 2017). An addition to the perturbation by Meignan and Knust (2019) is to only accept moves that at least perturb a part of the solution that causes a soft constraint violation. Furthermore, Goh et al. (2022) apply perturbation moves until the solution quality has been changed by at least 5%.

Other methods apply a ruin-and-recreate framework to both perturb and improve the solution (Abdelghany et al., 2021a; Rahimian et al., 2017). Rahimian et al. (2017) apply a ruin-and-recreate framework where a part of the solution is removed, and recreated using an IP solver. The choice of which part of the solution is destroyed was made probabilistically, based on how much was contributed to the total solution penalty. Each time the ruin-and-recreate was applied, either several nurses, days, weeks or cells were selected. Abdelghany et al. (2021a) randomly chose several nurses to destroy their rosters, and recreated them using a dynamic programming approach.

4 Problem description

In this chapter, we give a description of the problem to be addressed in this study. In Section 1.2, a general description of the NRP was given. Here, we further specify the objective function and constraints considered in the target applications.

4.1. Definition of consecutive shifts

Several constraints concern consecutive shifts, and thus a definition for consecutiveness of shifts is required. However, such a definition is not provided in the Dutch Working Hours Act, from which most hard constraints originate. In this problem, we consider two shifts to be consecutive if and only if there is at most 32 hours between the end of the first shift and the beginning of the second shift, and there is no other shift in between. This definition is used for consistency with other methods used within ORTEC, and is related to the weekly rest constraint described in Section 4.2, where a rest period of 32 hours is the shortest possible period that can contribute to satisfying the constraint.

4.2. Hard constraints

First, we consider the hard constraints, i.e. those constraints that determine whether a solution is feasible.

Dutch Working Hours Act The main hard constraints are the rules that come from the Dutch Working Hours Act (Arbeidstijdenwet, 2022, §5.2). A complete list of the rules that are considered can be found in Appendix B. Note that rules about breaks and shift lengths are assumed to be incorporated in the definitions of the shift types, which are considered as fixed input in this problem. By a cutting plane reduction, it was previously determined at ORTEC that in most cases in practice, all rules from the working hours act are satisfied, if the subset of rules listed below is satisfied. Therefore, only these rules are taken into account in the model. Feasible solutions that result from applying an algorithm to the model could then afterwards be checked if they also satisfy the remaining rules. Note that we do not do this in this study, because of the additional time required to implement the remaining rules.

- **Daily rest:** in every 24-hour period beginning at the start of an assigned shift, an employee must have a minimum of 11 hours of consecutive rest. An exception to this rule can be made once every 7 days, where the minimum is lowered to 8 hours.
- **Weekly rest:** an employee must have a minimum consecutive rest time of either
  - 36 hours every 7-day period beginning at the start of an assigned shift, or
  - 72 hours every 14-day period beginning at the start of an assigned shift. In this case, the rest time may be split into two periods of at least 32 hours each.
- **Working Sundays:** in every 52-week period, an employee must have a minimum of 13 non-working Sundays. For a scheduling period of a month, the allowed number of working Sundays is determined by the

number of worked Sundays in the previous 11 months. Note that to check if an employee works on a Sunday, we need to consider both shifts that start on Sunday, and shifts that start on the Saturday before and spill over to Sunday.

- **Night shift related:** a working shift is considered a 'night shift' if at least 1 hour is worked between the hours of midnight and 6 am. The following rules apply specifically to night shifts:
  - Following a sequence of at least 3 consecutive night shifts, an employee must have a minimum rest time of 46 consecutive hours.
  - If a sequence of consecutive shifts for an employee contains at least one night shift, then that sequence may contain at most 7 shifts.
  - An employee is allowed to work at most 36 night shifts that end after 2 am in every 16-week period. For a scheduling period of a month, the allowed number of night shifts is determined by the number of worked night shifts in the previous 12 weeks.

Maximum workload Each employee has an individual contract, which includes their contractual working hours per time period. This time period is typically a year, and the monthly workload limit used in our target application is based on it. To maintain some flexibility for the planners and employees, a buffer of 10 hours is added to the employees' workload limits, such that they can work some overtime if necessary. It is considered a hard constraint that employees should not work more than this increased workload limit.

Required skill levels Each employee has an individual set of skills, and each shift type can have one or more skill requirements. A roster is only considered feasible if no employees are assigned to shifts for which they lack the required skills. Note that in our problem, the skill types are incorporated into the shift types, such that a shift type is defined by the start and end times and the used skills.

Fixed assignments Employees can have predefined fixed assignments to shifts or days off. It is not allowed to alter a fixed assignment. Note that if an employee has a fixed day off on day d, no shift may be assigned to that same employee that starts on day $d - 1$ and ends on day d.

4.3. Soft constraints

The soft constraints considered in the model are listed below. For the description of the corresponding penalty calculations, we introduce the used notation: the rostering period is defined by the set of employees, E, the set of days, D, and the set of shift types, S. The binary decision variables are denoted by $x_{e,d,s}$: $x_{e,d,s}$ equals 1, if employee $e \in E$ is assigned to shift type $s \in S$ on day $d \in D$, and 0 otherwise. We denote a day-off assignment by o, and let $x_{e,d,o}$ be equal to 1 if employee e is assigned to a day off on day d, and 0 otherwise. An overview of the notation is given in Tables C.1 and C.2.

Shift coverage To fulfill the work demand, shift coverage constraints provide a minimum number of employees required for each shift type on each day. A distinction is made between regular and priority shifts by the height of the penalty they cause if they remain unplanned. Given the coverage requirement of shift type $s \in S$ on day $d \in D$ in number of employees, denoted by input parameter $a_{d,s}$, we calculate the corresponding coverage shortage, denoted by auxiliary variable $y_{d,s}$, as follows: we count the number of employees that are assigned to shift type s on day d, i.e. we sum over the values of $x_{e,d,s}$ for all employees $e \in E$. This number is then subtracted from the required number of employees, $a_{d,s}$. Because coverage surpluses are not penalized or rewarded, we let $y_{d,s}$ be equal to the maximum of this difference and zero. The given penalty for shift type s on day d is then proportional to the coverage shortage $y_{d,s}$. The weight of the penalty is 500 if shift type s is a priority shift type, and 100 otherwise. This is indicated by binary input parameter β_s, which is 1 if shift type s is a priority shift type, and 0 otherwise. By summing these penalties over all days $d \in D$ and shift types $s \in S$, we obtain the total coverage penalty, as shown in Table 4.2.

Shift coverage spread To spread coverage shortages evenly over the scheduling period, an additional penalty is given, that is greater if the same amount of undercoverage is distributed less evenly over the days. The penalty is only given if the undercoverage on a day is greater than 1, because a single shift cannot be distributed more evenly over different days. We let y_d denote the sum of the coverage shortages $y_{d,s}$ of all shift types $s \in S$ on day

$d \in D$. The coverage spread penalty is then illustrated as follows: if y_d is 1, the penalty equals 0. If y_d is 2, the penalty equals 100, if y_d is 3, the penalty equals 100 plus 200, i.e. 300. If y_d is 4, the penalty equals 100 plus 200 plus 300, i.e. 600, etc. The total coverage spread penalty is then calculated as shown in Table 4.2.

Overtime hours spread To prevent an unfair division of the shifts over the employees, a penalty is given for each employee for the amount of worked overtime. The penalty is quadratic in the number of hours of overtime, such that the same amount of overtime, but distributed less evenly over the employees, results in a higher penalty. The durations of shifts of different types are given by input parameter f_s for all shift types $s \in S$, and the contractual working hours of the employees by c_e^{con} for all employees $e \in E$. The worked overtime of employee $e \in E$, denoted by auxiliary variable u_e, is then calculated as the contractual working hours c_e^{con}, subtracted from the sum of the durations f_s of all shifts worked by employee e, over all days $d \in D$ and shift types $s \in S$, as given by decision variables $x_{e,d,s}$. Because we do not reward negative overtime, we let u_e be equal to the maximum of this difference and zero. The total overtime hours spread penalty is then calculated as the sum of the squared overtime hours u_e over all employees $e \in E$, as shown in Table 4.2.

Employee preferences Employee preferences are incorporated into the model by soft constraints for day-on/off requests, shift-on/off requests and preferred shift and day-off sequence lengths, as described below. In view of fairness among the employees, the weights of these constraints are determined for each employee both by the number of requests and the contractual working hours. This is done such that the sum of the weights of all preference constraints is higher for employees who have a higher number of contractual working hours, and this total weight of an employee is evenly distributed over all its preference constraints. Thus, the same weight j_e is given to all requests and the preferred sequence length constraints of employee $e \in E$. The total weights of employees, based on the contractual working hours per week, are shown in Table 4.1. As an example, suppose that an employee has 30 contractual working hours per week, one day-off request, one shift-on request, and preferred shift and day-off sequence lengths. Then, this employee receives a total weight of 80. The total number of constraints for this employee is four, so each request and preferred sequence length constraint receives a weight j_e of 20 (i.e. $\frac{80}{4}$).

- **Day-on/off requests:** employees can request to work or have a day off on specific days in the planning horizon. A penalty is given for each request that is not fulfilled. The requests are indicated by binary input parameters $g_{e,d}^{on}$ and $g_{e,d}^{off}$, which are 1 if employee $e \in E$ has requested to work or have a day off on day $d \in D$, respectively, and 0 otherwise. A penalty of weight j_e is given if employee e has requested to work on day d, but is assigned to a day off, i.e. $g_{e,d}^{on}$ equals 1 and $x_{e,d,o}$ equals 1, or if employee e has requested a day off on day d, but is not assigned to a day off, i.e. $g_{e,d}^{off}$ equals 1 and $x_{e,d,o}$ equals 0. The sum of all day requests penalties is shown in Table 4.2.

- **Shift-on/off request:** employees can request to work or not having to work specific shift types on specific days. A penalty is given for each request that is not fulfilled. The requests are indicated by binary input parameters $h_{e,d,s}^{on}$ and $h_{e,d,s}^{off}$, which are 1 if employee $e \in E$ has requested to work or not work shift type $s \in S$ on day $d \in D$, respectively, and 0 otherwise. A penalty of weight j_e is given if employee e has requested to work shift type s on day d, but is not assigned to it, i.e. $h_{e,d,s}^{on}$ equals 1 and $x_{e,d,s}$ equals 0, or if employee e has requested not to work shift type s, but is assigned to it, i.e. $h_{e,d,s}^{off}$ equals 1 and $x_{e,d,s}$ equals 1. The sum of all shift requests penalties is shown in Table 4.2.

- **Preferred shift sequence length:** employees can specify that they prefer to work sequences of a specific number of consecutive shifts. A penalty is given that is proportional to the fraction of sequences of consecutive shifts that are not of the preferred length. For each employee $e \in E$, the sequences of consecutive shifts and their lengths are determined based on the shift assignments, specified by decision variables $x_{e,d,s}$, the shift start and end times, and the definition of consecutive shifts, as described in Section 4.1. We then let auxiliary variables η_e^{on} be equal to the number of sequences of consecutive shifts in the roster of employee e. Auxiliary variables ζ_e^{on} indicate the number of these sequences of consecutive shifts that are of the length that was preferred by employee e. Then, the fraction of sequences of consecutive shifts that are *not* of the preferred length is $\left(1 - \frac{\zeta_e^{on}}{\eta_e^{on}}\right)$. The penalty given for employee e is then equal to this fraction multiplied by the weight j_e for employee e.

  For example, suppose that employee $e \in E$ prefers to have sequences of consecutive shifts of length 2, and that their roster contains three sequences of consecutive shifts: one of length 1, one of length 2 and one of length 3. Then, the number of sequences of consecutive shifts of the preferred length is 1, i.e. $\zeta_e^{on} = 1$, and the total number of sequences of consecutive shifts is 3, i.e. $\eta_e^{on} = 3$. Then, the fraction of sequences of consecutive shifts that are *not* of the preferred length is $\left(1 - \frac{\zeta_e^{on}}{\eta_e^{on}}\right) = \left(1 - \frac{1}{3}\right) = \left(\frac{2}{3}\right)$. The corresponding penalty is equal to this fraction, multiplied by the preference constraints weight for employee e, j_e: $j_e\left(\frac{2}{3}\right)$. The total preferred shift sequence length penalty is shown in Table 4.2.

- **Preferred day-off sequence length:** employees can specify that they prefer sequences of consecutive days off to have a specific length. A penalty is given that is proportional to the fraction of sequences of consecutive days off that are not of the preferred length, similar to the penalty related to the preferred number of consecutive shifts, as described above. For each employee $e \in E$, the sequences of consecutive days off and their lengths are determined based on the shift assignments, specified by decision variables $x_{e,d,s}$, the shift start and end times, and the definition of consecutive shifts, as described in Section 4.1. The length of a sequence of consecutive days off is calculated as the number of hours between two shifts, divided by 24 hours and rounded to the nearest integer. Note that such a sequence is only counted if the time between two shifts is more than 32 hours, as otherwise it is only the rest between two consecutive shifts. We then let auxiliary variables η_e^{off} be equal to the number of sequences of consecutive days off in the roster of employee e. Auxiliary variables ζ_e^{off} indicate the number of these sequences of consecutive days off that are of the length that was preferred by employee e. Then, the fraction of sequences of consecutive days off that are *not* of the preferred length is $\left(1 - \frac{\zeta_e^{\mathit{off}}}{\eta_e^{\mathit{off}}}\right)$. The penalty given for employee e is then equal to this fraction multiplied by the weight j_e. The total preferred day-off sequence length penalty is shown in Table 4.2.

Table 4.1: This table shows the total weight of all preference constraints of an employee, i.e. the day/shift-on/off requests and the constraints for preferred shift and day-off sequence lengths, based on the contractual working hours of that employee.

Contractual working hours λ per week	**Total weight**
$\lambda \geq 32$	100
$32 > \lambda \geq 24$	80
$24 > \lambda \geq 16$	60
$16 > \lambda$	40

4.4. Objective function

The objective function is to minimize the sum of all soft constraint penalties. The soft constraint penalty calculations are shown in Table 4.2.

Table 4.2: Soft constraint penalty calculations, as described in Section 4.3. The used notation is clarified in Tables C.1 and C.2.

Soft constraint	**Penalty**
Shift coverage	$\sum_{d\in D}\sum_{s\in S} y_{d,s}\left(500b_s + 100(1-b_s)\right)$
Shift coverage spread	$\sum_{d\in D}\sum_{m=2}^{y_d} 100(m-1)$
Overtime hours spread	$\sum_{e\in E} u_e^2$
Day-on/off requests	$\sum_{e\in E}\sum_{d\in D} j_e\left(x_{e,d,o}g^{on}_{e,d} + (1-x_{e,d,o})g^{off}_{e,d}\right)$
Shift-on/off requests	$\sum_{e\in E}\sum_{d\in D}\sum_{s\in S} j_e\left((1-x_{e,d,s})h^{on}_{e,d,s} + x_{e,d,s}h^{off}_{e,d,s}\right)$
Preferred shift sequence length	$\begin{cases}\sum_{e\in E} j_e\left(1-\frac{\zeta_e^{on}}{\eta_e^{on}}\right), & \eta_e^{on} > 0\\ 0, & \eta_e^{on} = 0\end{cases}$
Preferred day-off sequence length	$\begin{cases}\sum_{e\in E} j_e\left(1-\frac{\zeta_e^{off}}{\eta_e^{off}}\right), & \eta_e^{off} > 0\\ 0, & \eta_e^{off} = 0\end{cases}$

5 Solution methods

Following the conclusion drawn in Section 2.4 and the categorization of algorithm components in Chapter 3, the research methodology is to first build and test different algorithm components, tailored to the problem formulation, and then evaluate the performance of different combinations/configurations of those components. The different solution methods that are implemented are described in this chapter.

5.1. Construction methods

The purpose of construction methods is to generate an initial feasible solution that is of sufficient quality for the subsequent overall framework to function properly. A construction method starts from an empty roster, where no shifts are assigned to employees, and builds up the initial solution by assigning shifts one by one. Since the coverage requirements are by far the most important soft constraints in the problem formulation, it is a logical construction approach to try to satisfy these as much as possible. However, since all hard constraints are specific to the employees, a construction method that is oriented on (required) shifts might not be able to assign the same number of shifts as a method that is oriented firstly on the employees. In this study, we consider two different approaches for construction methods, which reflect these different orientations, and which are based on the methods encountered in literature, see Section 3.2.

5.1.1. Construction-per-shift

The first approach is to list all shifts that arise from the coverage requirements, and then assign as many of those shifts to employees as possible, while maintaining the feasibility of the solution. In general, this method finishes with a set of shifts that cannot be assigned anymore without violating the hard constraints. The feasibility of assigning an additional shift to an employee depends on the previously assigned shifts. Therefore, the order in which the shifts are considered determines which shifts can be assigned by this method. In order to maximize the number of shifts that are assigned, the shifts are sorted beforehand based on different criteria, see Section 5.1.3. The idea is to try to assign the 'most difficult' shifts first. For each shift to be assigned, we go through all employees and try to assign it to employees until either we have reached the coverage requirement of the shift, or we have gone through all employees. A shift is assigned to a considered employee if it does not cause any hard constraint violation. If a shift is assigned to none of the employees, the shift is left unassigned, and we move to the next shift. Similar to the ordering of the shifts, the order in which the employees are considered also determines which shifts can be assigned. Thus, also the employees are sorted based on several criteria, see Section 5.1.3. Note that for each shift, only those employees are considered that are available, i.e. those that have the required skills for the shift and do not already have a fixed shift or day-off assignment on the corresponding day. The pseudocode for this approach is shown in Algorithm 1.

5.1.2. Construction-per-employee

In the second approach, feasible rosters are constructed for all employees separately, which are then combined into a complete roster. For each employee, we go through the days of the scheduling period from start to end. For each day, we collect the valid shifts, i.e. those shifts for which the considered employee has the required skills, and for which the coverage requirement on that day is at least one. This last condition is to ensure that employees are not assigned to shifts which are not required at all on the considered day. The valid shifts are then sorted

Algorithm 1 Construction-per-shift

```
Input: empty roster
 1: Collect requiredShifts from coverage requirements for roster
 2: Set shiftSortingCriteria and employeeSortingCriteria
 3: Sort requiredShifts by shiftSortingCriteria
 4: for each shift in requiredShifts do
 5:     Collect availableEmployees for shift
 6:     Sort availableEmployees by employeeSortingCriteria
 7:     Set currentCoverage = 0, employeeIndex = 0
 8:     while currentCoverage < requiredCoverage AND employeeIndex < numberOfAvailableEmployees do
 9:         employee = availableEmployees[employeeIndex]
10:         if employee is not yet assigned to a shift on day shift.day then
11:             Assign shift to employee
12:             if roster is still feasible then
13:                 currentCoverage++
14:             else
15:                 Unassign shift from employee
16:             end if
17:         end if
18:         employeeIndex++
19:     end while
20: end for
Output: constructed roster
```

based on several sorting criteria, see Section 5.1.3. We then go through all valid shifts and try to assign them to the employee until either we have assigned one of the shifts, or we have gone through all valid shifts. A shift is assigned to the considered employee if it does not cause any hard constraint violation. If none of the valid shifts is assigned, the considered day is left as a day off for the employee, and we move to the next day. To also take the coverage requirements into account in this construction method, one of the sorting criteria for the valid shifts is the difference between the required coverage and the coverage achieved by the employee rosters that are already constructed. The resulting combined roster is then affected by the order in which the employee rosters are constructed. Therefore, we sort the employees beforehand, based on criteria explained in Section 5.1.3. The pseudocode for this approach is shown in Algorithm 2.

5.1.3. Sorting criteria

Both construction methods described above apply sorting of shifts and employees. This is done based on several criteria that indicate either how difficult a shift or employee is to assign, such as the number of available employees for a shift, or how beneficial an assignment is to the roster penalty, such as the current shift coverage or day-off requests. The different criteria are applied hierarchically, such that the objects are sorted first by the most important criterion. Those objects that are equal in this first criterion are then sorted by the second most important criterion, etc., up to the last used criterion. Objects that are equal in the final criterion are sorted randomly. The hierarchy of the sorting criteria thus determines the ordering of the objects.

Shift sorting criteria The criteria used for sorting shifts are listed below.

- **Number of available employees:** available employees for a shift are those employees that have the required skills and who do not have a fixed shift or day-off assignment. A shift with fewer available employees is given a higher priority, in order to reduce the probability that all available employees are already assigned to another shift, once we try to assign this shift.
- **Night shifts:** a higher priority is given to night shifts, since they are specifically involved in several hard constraints, which makes it more difficult to assign them later on in the process.
- **Sunday shifts:** the number of working Sundays per employee is restricted by a hard constraint, which makes Sunday shifts more difficult to assign when more shifts are already assigned. Thus, Sunday shifts

Algorithm 2 Construction-per-employee

```
Input: empty roster
 1: Set shiftSortingCriteria and employeeSortingCriteria
 2: Sort employees in roster by employeeSortingCriteria
 3: for each employee in employees do
 4:     for each day in days do
 5:         Collect validShifts for employee on day
 6:         Sort validShifts by shiftSortingCriteria
 7:         Set shiftIndex = 0, shiftAssigned = false
 8:         while shiftIndex < numberOfValidShifts AND shiftAssigned = false do
 9:             shift = validShifts[shiftIndex]
10:             Assign shift to employee
11:             if roster is still feasible then
12:                 shiftAssigned = true
13:             else
14:                 Unassign shift from employee
15:                 shiftIndex++
16:             end if
17:         end while
18:     end for
19: end for
Output: constructed roster
```

are given a higher priority. Note that a Sunday shift can be either a shift that starts on Sunday or a shift that starts on Saturday and ends on Sunday.

- **Priority shifts:** priority shifts that remain unplanned result in a higher penalty than unplanned regular shifts. Therefore, priority shifts are considered before regular shifts.
- **Date:** shifts which are earlier in the scheduling period are given a higher priority, because they are affected by the planning history the most, which might make them harder to assign.
- **Coverage requirements:** shifts for which more employees are required are given a higher priority. Note that for Construction-per-employee, the required number of employees of a shift is recalculated after each completed employee roster, based on the employees that are already assigned.
- **Shift-on/off requests:** satisfying shift-on/off requests reduces the roster penalty. Therefore, for an employee, a shift with a shift-on or a shift-off request is given a higher or a lower priority, respectively. Note that this criterion can only be used in Construction-per-employee.

Employee sorting criteria The criteria used for sorting employees are listed below. Criteria marked with ** are only relevant for Construction-per-shift.

- **Number of valid shift types:** valid shift types are those shift types for which an employee has the required skills. Employees that have fewer valid shift types are given a higher priority, to reduce the probability that later on there are no shifts left anymore that can be assigned to them.
- **Available working time:** the more shifts an employee has to work, the harder it is to assign them such that they do not violate any hard constraints. Therefore, a higher priority is given to employees with more available working hours. Note that for Construction-per-shift, the available working time is recalculated after each assigned shift, such that for each shift, this criterion becomes the available working time left.
- **Shift-on/off requests**:** satisfying shift-on/off requests reduces the roster penalty. Therefore, for a shift, an employee with a shift-on or a shift-off request for the same shift is given a higher or a lower priority, respectively.
- **Day-on/off requests**:** satisfying day-on/off requests reduces the roster penalty. Therefore, for a shift, an employee with a day-on or a day-off request for the same day is given a higher or a lower priority, respectively.

5.2. Neighborhood structures

Both overall frameworks that we apply, further explained in Section 5.3, use a set of neighborhood structures. A neighborhood structure is characterized by a specific type of operation, e.g. swapping the shifts of two employees on the same day. The neighborhood of a current solution is then defined as the set of solutions that can be reached by applying the corresponding operation exactly once, somewhere in the current solution. The different neighborhood structures that we use are listed below, and Figures 5.1-5.6 show examples of the corresponding operations. Abbreviations for the neighborhood structures are shown in Table 5.1.

Vertical swapping neighborhood structures (Vx) The vertical swapping operation is applied to a pair of employees on x consecutive days, where x is a property of the neighborhood. This property is called the block length and indicates the number of consecutive days for which the shift assignments of the two employees are swapped. Thus, vertical swapping operations with different block lengths result in separate neighborhood structures, such that a neighborhood structure Vx only contains vertical swapping operations with block length x. Note that within this operation, a day-off assignment is swapped in the same way as a regular shift assignment. Examples of vertical swapping operations are shown in Figure 5.1.

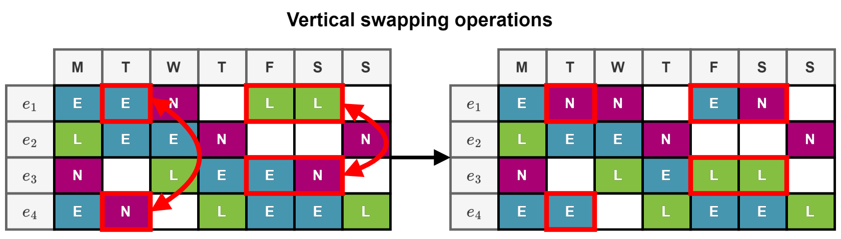


Figure 5.1: Examples of vertical swapping operations with block lengths 1 and 2: swapping the early shift E on Tuesday for employee e_1 with the night shift N on the same day for employee e_4, and swapping the late shifts L on Friday and Saturday for employee e_1 with the E and N shifts on the same days for employee e_3.

Horizontal swapping neighborhood structures (Hx) The horizontal swapping operation swaps the shifts for one employee of two non-overlapping blocks of x consecutive days. Similar to the vertical swapping neighborhood structures, different block lengths result in separate horizontal swapping neighborhood structures, and day-off assignments may be part of swapping operations. Examples of horizontal swapping operations are shown in Figure 5.2.

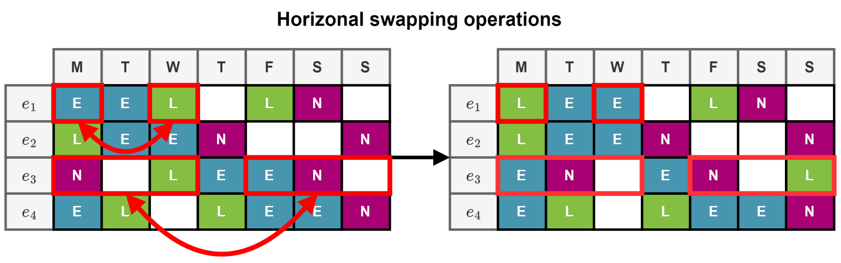


Figure 5.2: Examples of horizontal swapping operations with block lengths 1 and 3: swapping the early shift E on Monday for employee e_1 with the late shift L on Wednesday for the same employee, and swapping the shift and day-off assignments of Monday up to Wednesday for employee e_3 with those of Friday up to Sunday for the same employee.

Change neighborhood structures (Cx) The change operation changes the shift assignments for an employee on a block of x consecutive days, by unassigning the currently assigned shifts, and replacing them by other shifts on the same days. For the change neighborhood structure, a day-off is considered as a possible shift assignment, such that changing a day-off to any other shift assignment, and changing any shift assignment to a day-off are also part of this neighborhood structure. To reduce the size of this neighborhood for block lengths larger than 1, the same shift assignment is applied to all days in the block. Thus, for example, a change operation with block length 3 can assign three consecutive day shifts, but not two day shifts followed by a night shift. Similar to the swapping neighborhood structures, different block lengths result in separate change neighborhood structures. Examples of change operations are shown in Figure 5.3.

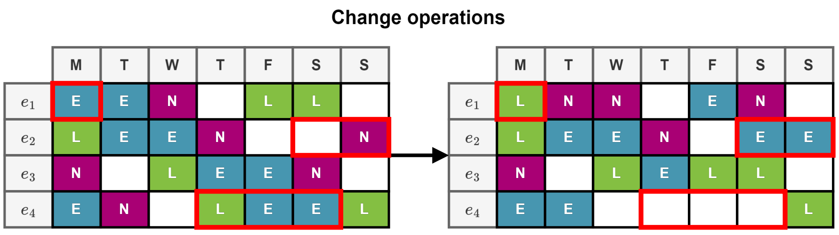


Figure 5.3: Examples of change operations with block lengths 1, 2 and 3: changing the early shift assignment of employee e_1 on Monday into a late shift assignment, the day-off and night shift assignment of employee e_2 on Saturday and Sunday into two early shift assignments, and the late and early shift assignments of employee e_4 on Thursday, Friday and Saturday into three day-off assignments.

Coverage-focused neighborhood structures Coverage requirements are the most important soft constraints in our problem. Therefore, we adopt the following three coverage-focused neighborhoods, of which UC1 and UC2 are the same as those used by Abdelghany et al. (2021a, 2021b). These neighborhood structures require the generation of a list of currently undercovered shifts, before they can be applied. Note that these neighborhoods are special cases of the change neighborhood structure.

1. **UC1:** the operation of the first neighborhood structure is to assign a currently undercovered shift to an employee that currently has a day-off on the corresponding day. An example of a UC1 operation is shown in Figure 5.4.

2. **UC2:** for the second neighborhood structure, the operation is to assign a currently undercovered shift to an employee that currently has a day-off on the corresponding day, and simultaneously unassign any of the shifts for the same employee on another day. An example of a UC2 operation is shown in Figure 5.5.

3. **UC3:** for the third neighborhood structure, the operation is to assign a currently undercovered shift to an employee who is already assigned to another shift on the same day. Thus, the previously assigned shift is unassigned. This operation is very similar to that of UC2, but differs in that it replaces the undercovered shift for a shift on the same day, instead of one on another day. An example of a UC3 operation is shown in Figure 5.6.

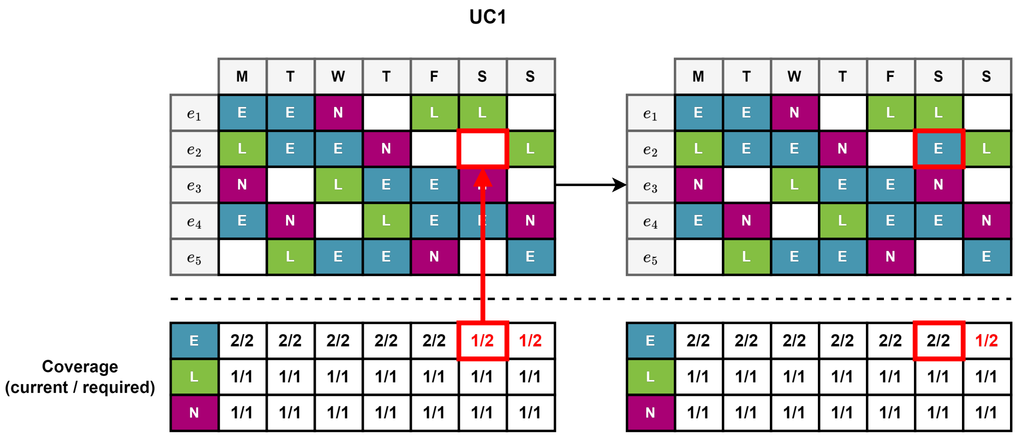
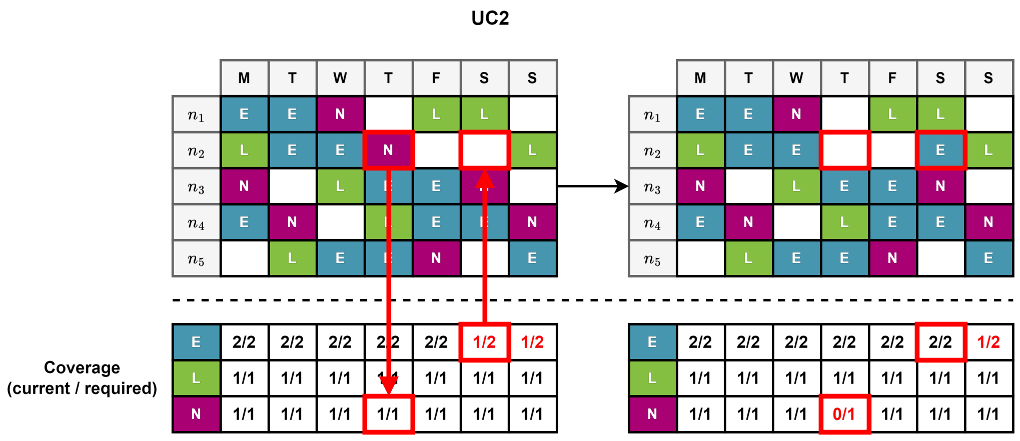

UC1
M T W T F S S
e_1 E E N L L
e_2 L E E N L
e_3 N L E E N
e_4 E N L E E N
e_5 L E E N E
M T W T F S S
e_1 E E N L L
e_2 L E E N E L
e_3 N L E E N
e_4 E N L E E N
e_5 L E E N E
Coverage (current / required)
E 2/2 2/2 2/2 2/2 2/2 1/2 1/2
L 1/1 1/1 1/1 1/1 1/1 1/1 1/1
N 1/1 1/1 1/1 1/1 1/1 1/1 1/1
E 2/2 2/2 2/2 2/2 2/2 2/2 1/2
L 1/1 1/1 1/1 1/1 1/1 1/1 1/1
N 1/1 1/1 1/1 1/1 1/1 1/1 1/1

Figure 5.4: Example of a UC1 operation. The undercovered early shift on Saturday is assigned to employee e_2.

Neighborhood restrictions The neighborhood structures described above are defined by their characterizing operations. However, the neighborhoods for a given solution are restricted further by several criteria:

- **Identical neighbors:** a neighbor is not considered if it is identical to the current solution, for example, if it is formed by swapping two identical shift assignments between two employees.



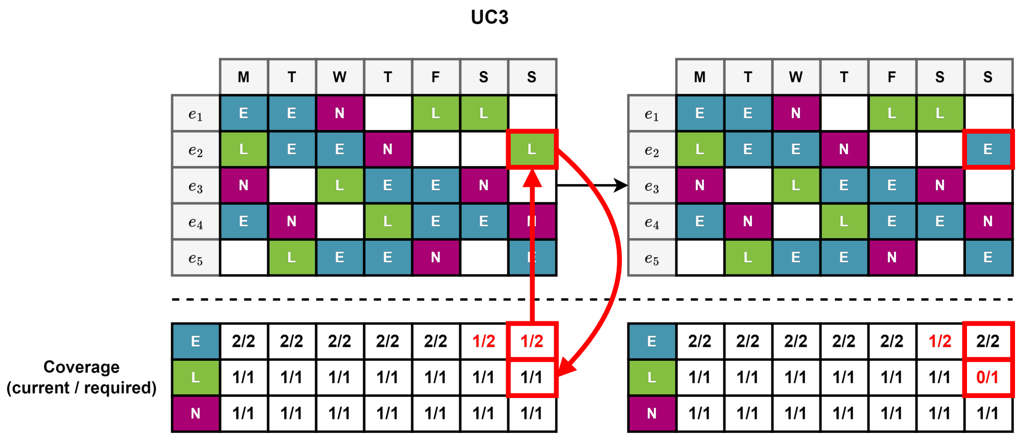
	M	T	W	T	F	S	S
n_1	E	E	N		L	L	
n_2	L	E	E	N			L
n_3	N		L	E	E	N	
n_4	E	N		L	E	E	N
n_5		L	E	E	N		E

	M	T	W	T	F	S	S
E	2/2	2/2	2/2	2/2	2/2	1/2	1/2
L	1/1	1/1	1/1	1/1	1/1	1/1	1/1
N	1/1	1/1	1/1	1/1	1/1	1/1	1/1

	M	T	W	T	F	S	S
n_1	E	E	N		L	L	
n_2	L	E	E			E	L
n_3	N		L	E	E	N	
n_4	E	N		L	E	E	N
n_5		L	E	E	N		E

	M	T	W	T	F	S	S
E	2/2	2/2	2/2	2/2	2/2	2/2	1/2
L	1/1	1/1	1/1	1/1	1/1	1/1	1/1
N	1/1	1/1	1/1	0/1	1/1	1/1	1/1

Figure 5.5: Example of a UC2 operation. The undercovered early shift on Saturday is assigned to employee e_2, while also the night shift of Thursday is unassigned for employee e_2.



	M	T	W	T	F	S	S
e_1	E	E	N		L	L	
e_2	L	E	E	N			L
e_3	N		L	E	E	N	
e_4	E	N		L	E	E	N
e_5		L	E	E	N		E

	M	T	W	T	F	S	S
E	2/2	2/2	2/2	2/2	2/2	1/2	1/2
L	1/1	1/1	1/1	1/1	1/1	1/1	1/1
N	1/1	1/1	1/1	1/1	1/1	1/1	1/1

	M	T	W	T	F	S	S
e_1	E	E	N		L	L	
e_2	L	E	E	N			E
e_3	N		L	E	E	N	
e_4	E	N		L	E	E	N
e_5		L	E	E	N		E

	M	T	W	T	F	S	S
E	2/2	2/2	2/2	2/2	2/2	1/2	2/2
L	1/1	1/1	1/1	1/1	1/1	1/1	0/1
N	1/1	1/1	1/1	1/1	1/1	1/1	1/1

Figure 5.6: Example of a UC3 operation. The late shift assigned to employee e_2 on Sunday is replaced by the undercovered early shift on the same day.

- **Required skills:** a neighbor is not considered if it contains an assignment of a shift to an employee who does not have the required skills.
- **Fixed assignments:** a neighbor is not considered if it involves altering a fixed shift or day-off assignment.
- **Non-required shifts:** a change or swap operation is not considered if it involves assigning a shift on a day on which no shift of that type is required, i.e. where the corresponding coverage requirement is equal to zero. This restriction was added because it was seen as undesirable that employees are assigned to shifts that do not exist in practice. Note that this restriction implicitly adds a hard constraint to the problem.
- **Equivalent neighbors:** for the swapping neighborhood structures with a block length greater than 1, operations where the first or the last shift assignment of the two blocks are identical are not considered, since they are equivalent to swaps of a smaller block length. Similarly, for the change neighborhood structures with a block length greater than 1, operations where the current first or last shift assignment of the block is already equal to the new shift assignment are not considered. In practice, swapping and change neighborhoods with smaller block lengths are always applied too. Therefore, this restriction reduces the number of duplicate neighbors considered in separate neighborhoods, without completely removing neighbors from consideration.

Table 5.1: Neighborhood structures with abbreviations

Neighborhood structure	**Abbreviation**
Vertical swapping with block length x	Vx
Horizontal swapping with block length x	Hx
Change operations with block length x	Cx
Assign undercovered shifts neighborhood structure	UC1
Replace undercovered shifts for shifts on other days	UC2
Replace undercovered shifts for shifts on the same day	UC3

5.3. Overall frameworks

The purpose of the overall framework of an algorithm is to improve the initial solution generated by the applied construction method, and output a final feasible solution that constitutes the result of the algorithm as a whole. The two overall frameworks used in this study, explained below, can be characterized as local search methods that apply different neighborhood structures in an attempt to reach a globally ‘good’ local optimum. This requires the ability to descend towards local optima, as well as the ability to escape them. Note that unless we already know an optimal solution, we cannot know how good a local optimum really is compared to the global optimum. The use of different neighborhood structures enables the frameworks to find better local optima, since a local optimum with respect to one neighborhood structure is not necessarily a local optimum with respect to another. A local optimum with respect to all applied neighborhood structures combined is then at least as good as the local optima with respect to any of the individual neighborhood structures on its own.

5.3.1. Variable Neighborhood Search

The Variable Neighborhood Search framework consists of two phases, namely Variable Neighborhood Descent and Perturbation, which are alternated repeatedly, until a specified time limit is reached. The Variable Neighborhood Descent steers the solution towards a local optimum, by sequentially going through the set of neighborhood structures in search of improving changes and swaps. Within this descent phase, a move is accepted only if the solution will remain feasible, and the roster penalty will be at least as good as before. The different neighbors within a neighborhood are considered in a random order. If, after going through a neighborhood completely, at least one improving move is found, we go back to the first neighborhood, otherwise we go to the next one. This is repeated until we have gone through all neighborhoods without having found an improving move, meaning that a local optimum with respect to the combined set of neighborhoods has been reached. The purpose of the following perturbation phase is then to escape the local optimum, by relocating to a different point in the solution space, from which the next descent phase can potentially lead to a different local optimum, which might be better than the previous one. Here, moves are chosen from a set of neighborhoods at random, and are accepted independent of their effect on the roster penalty, as long as the solution will remain feasible. This is done until a limit is reached of a minimum number of accepted moves and/or a minimum amount of relative change in the roster penalty. Note that the sets of neighborhoods used in Variable Neighborhood Descent and Perturbation do not have to be equal. The pseudocode for Variable Neighborhood Search is shown in Algorithms 3, 4 and 5.

Algorithm 3 Variable Neighborhood Search

```
Input: initial solution $x \in X$ and objective function $f : X \to \mathbb{R}$ for solution space $X$
1: Initialize $x_{best}$ as $x$
2: Set end time $t^{end}$
3: Set $t$ to current time
4: while $t < t^{end}$ do
5:     Variable Neighborhood Descent
6:     Perturbation
7:     Set $t$ to current time
8: end while
Output: $x_{best}$
```

Algorithm 4 Variable Neighborhood Descent

```
Input: current solutions $x$ and $x_{best}$, end time $t^{end}$ and objective function $f$
 1: Define search neighborhoods $N_1^{search}, ..., N_{k^{max}}^{search}$
 2: Set $k = 1$
 3: Set $t$ to current time
 4: while $k \leq k^{max}$ AND $t < t^{end}$ do
 5:     Set $improvingMoveFound = false$
 6:     Generate neighbors in $N_k^{search}$
 7:     Set $t$ to current time
 8:     while neighbors left in $N_k^{search}$ AND $t < t^{end}$ do
 9:         Select next neighbor $x'$ randomly from $N_k^{search}$
10:         if $x'$ is feasible AND $f(x') \leq f(x)$ then
11:             if $f(x') < f(x)$ then
12:                 Set $improvingMoveFound = true$
13:             end if
14:             Set $x = x'$
15:         end if
16:         Remove $x'$ from $N_k^{search}$
17:         Set $t$ to current time
18:     end while
19:     if $f(x) < f(x_{best})$ then
20:         Set $x_{best} = x$
21:     end if
22:     if $improvingMoveFound = true$ then
23:         Set $k = 1$
24:     else
25:         $k++$
26:     end if
27:     Set $t$ to current time
28: end while
Output: improved solutions $x$ and $x_{best}$
```

Algorithm 5 Perturbation

```
Input: current solutions $x$ and $x_{best}$, end time $t^{end}$ and objective function $f$
 1: Define perturbation neighborhoods $N_1^{perturbation}, ..., N_{q^{max}}^{perturbation}$
 2: Set perturbation penalty limit $L_P$ and perturbation moves limit $L_M$
 3: Set starting penalty $P_{start} = f(x)$ and perturbation moves counter $movesCounter = 0$
 4: while $|f(x) - P_{start}| < L_P$ AND $movesCounter < L_M$ AND $t < t^{end}$ do
 5:     Select $q \in \{1, ..., q^{max}\}$ randomly
 6:     Select random solution $x'$ from $N_q^{perturbation}$
 7:     if $x'$ is feasible then
 8:         Set $x = x'$
 9:         if $f(x) < f(x_{best})$ then
10:             Set $x_{best} = x$
11:         end if
12:         $movesCounter++$
13:     end if
14:     Set $t$ to current time
15: end while
Output: updated solutions $x$ and $x_{best}$
```

5.3.2. Simulated Annealing

The idea of the Simulated Annealing framework is to try to improve the solution over many iterations, where in each iteration a move is randomly chosen from one of the neighborhood structures, and the choice to accept it or not is made probabilistically based on its effect on the roster penalty, while maintaining roster feasibility. Moves towards infeasible neighbors are rejected. If a move towards a feasible neighbor improves the roster, it is always accepted. For moves towards feasible neighbors that increase the roster penalty, a larger increase in the penalty results in a lower probability of acceptance. The acceptance probability is given by $P(\Delta) = e^{-\Delta/T}$, where Δ is equal to the difference between the new penalty and the current penalty, and T is a parameter known as the temperature. Furthermore, the probability of accepting solution worsening moves decreases over time, such that the algorithm starts with a wide search across the solution space, easily escaping local minima, and finally converges to a single local minimum, in which it can descend with more and more precision. This behavior is achieved by applying a so-called cooling schedule, that decreases the temperature T over time. We use the same cooling schedule as Ceschia et al. (2023), where the temperature is decreased by a factor α, known as the cooling ratio, every time either a number M_s of moves have been sampled, or a number M_a of moves have been accepted, whichever comes first. By choosing M_a to be a fraction of M_s, the temperature is decreased faster when many moves are accepted. Effectively, this mostly speeds up the start of the algorithm, when many moves are accepted due to the high temperature and relative abundance of improving moves. The pseudocode for the Simulated Annealing framework is shown in Algorithm 6.

Algorithm 6 Simulated Annealing

```
Input: initial solution $x \in X$ and objective function $f : X \to \mathbb{R}$ for solution space $X$
 1: Initialize $x_{best}$ as $x$
 2: Set temperature $T$, cooling rate $\alpha$, temperature limit $T^{min}$, and end time $t^{end}$
 3: Set limits $M_s$ and $M_a$ for the numbers of sampled and accepted neighbors before cooling
 4: Define neighborhoods $N_1, ..., N_k$
 5: Set $m_s = 0$ and $m_a = 0$
 6: Set $t$ to current time
 7: while $t < t^{end}$ AND $T > T^{min}$ do
 8:     Select neighborhood $N$ randomly from $N_1, ...N_k$
 9:     Select solution $x'$ randomly from $N$
10:     if $x'$ is feasible then
11:         if $f(x') \leq f(x)$ then
12:             Set $x = x'$
13:             if $f(x) < f(x_{best})$ then
14:                 Set $x_{best} = x$
15:             end if
16:             $m_a + +$
17:         else
18:             Set $\Delta = f(x') - f(x)$
19:             Calculate acceptance probability $P = e^{-\Delta/T}$
20:             Draw random number $r \in [0, 1]$
21:             if $r < P$ then
22:                 Set $x = x'$
23:                 $m_a + +$
24:             end if
25:         end if
26:     end if
27:     $m_s + +$
28:     if $m_s \geq M_s$ OR $m_a \geq M_a$ then
29:         Set $T = \alpha T$
30:         Set $m_s = 0$ and $m_a = 0$
31:     end if
32:     Set $t$ to current time
33: end while
Output: $x_{best}$
```

6 Evaluation methods

In this chapter, we elaborate on how the solution methods, described in Chapter 5, are evaluated. Overall, the strategy is to first find the best performing construction method configuration, which we then fix in order to evaluate the following overall frameworks and their variations. Secondly, different combinations of overall frameworks and construction methods are tested. Finally, the algorithms are tested with slightly modified problem formulations, to compare with lower bounds and ORTEC's algorithms.

6.1. Experimental setting

The solution methods are evaluated based on their performance in the setting of a Dutch hospital that is a client of ORTEC. Data was gathered from three different departments over the months April, May and June of 2023. Characteristics of the resulting nine instances are shown in Table 6.1. In the remainder of this report, the instances will be referred to by their abbreviations, as denoted in the same table. The only data that were not available for these instances were the preferred shift and day-off sequence lengths. Therefore, for the corresponding soft constraints, fictional preferences were generated, based on the contractual working hours of the employees, as shown in Table 6.2. All tests are run on a machine with an Intel Xeon Gold 3.20GHz processor and 8GB RAM. Tests including an overall framework are executed with a time limit of 10 minutes.

Table 6.1: Characteristics of the client instances: numbers of required shifts (S), priority shifts (PS), night shifts (NS), shift types (ST), employees (E), average maximum working hours per employee (MH), average required working hours per employee (RH), requests (R, sum of the following four request types), day-on requests (DOnR), day-off requests (DOffR), shift-on requests (SOnR) and shift-off requests (SOffR). Note that the required shifts include the priority and night shifts, and that the maximum and required working hours are counted for the whole month.

Dept.	Month	Abbr.	S	PS	NS	ST	E	MH	RH	R	DOnR	DOffR	SOnR	SOffR
Obstetrics	April	O4	951	80	242	15	85	88.0	86.7	186	76	88	22	0
	May	O5	992	86	252	15	85	92.1	93.7	104	86	18	0	0
	June	O6	956	92	240	15	84	90.4	91.4	98	85	13	0	0
Trauma	April	T4	429	36	60	10	38	114.0	90.8	108	27	62	6	13
	May	T5	447	40	62	10	37	120.7	97.2	154	41	90	10	13
	June	T6	438	44	60	10	36	116.2	97.7	65	11	40	0	14
Vascular surgery	April	V4	441	33	60	10	46	121.6	77.5	56	0	51	1	4
	May	V5	443	38	62	12	45	127.9	79.6	33	1	31	1	0
	June	V6	563	47	60	12	45	124.1	100.9	24	0	24	0	0

6.2. Construction methods

The first step is to determine the shift and employee sorting criteria and corresponding hierarchies that lead to the best performance of the construction methods. Because the total number of possibilities in terms of chosen

Table 6.2: This table shows the fictional preferred shift and day-off sequence lengths, based on the contractual working hours of that employees, as they were used in the problem instances.

Contractual working hours λ per week	**Preferred shift sequence length**	**Preferred day-off sequence length**
$\lambda \geq 32$	4	2
$32 > \lambda \geq 24$	3	3
$24 > \lambda \geq 16$	3	4
$16 > \lambda$	2	5

criteria and corresponding hierarchies is enormous, is it practically impossible to evaluate them all. Therefore, we form the sets of shift and employee sorting criteria and the corresponding hierarchies heuristically.

The candidate sorting criteria for shifts and employees, as described in Section 5.1.3, are placed in the sets $C_{\mathcal{S}}$ and $C_{\mathcal{E}}$, respectively. We let $\mathcal{S}$ and $\mathcal{E}$ denote the ordered sets of sorting criteria for shifts and employees, respectively, to be used in the construction methods shown in Algorithms 1 and 2. $\mathcal{S}$ and $\mathcal{E}$ are initialized as empty sets, which means that shifts and employees are sorted completely randomly in the construction method. The performance of the construction method in this setting is considered as the baseline. Criteria from $C_{\mathcal{S}}$ and $C_{\mathcal{E}}$ are then added iteratively to $\mathcal{S}$ and $\mathcal{E}$, based on their effect on the performance of the construction method. For this, for each criterion in $C_{\mathcal{S}}$ and $C_{\mathcal{E}}$ separately, we test the performance of the construction method after adding only this criterion to either $\mathcal{S}$ or $\mathcal{E}$. The criterion that results in the best performance, in terms of the objective function value, over most of the tested instances is then permanently added to either $\mathcal{S}$ or $\mathcal{E}$ and removed from $C_{\mathcal{S}}$ or $C_{\mathcal{E}}$. After adding this best criterion, the addition of the remaining criteria in $C_{\mathcal{S}}$ and $C_{\mathcal{E}}$ is tested in the same way, after which again the best criterion is added to either $\mathcal{S}$ or $\mathcal{E}$. This process is repeated until either both $C_{\mathcal{S}}$ and $C_{\mathcal{E}}$ are empty or the addition of none of the remaining criteria leads to better performance than already obtained in the previous iteration (or the initial baseline, in case of the first iteration). Pseudocode of this method is shown in Algorithm 7. Note that in each iteration, either a shift or an employee sorting criterion can be added. Furthermore, when a criterion is added to $\mathcal{S}$ or $\mathcal{E}$, it is placed lowest in the ordering, such that the criterion that is added first in either of the sets will be highest in the hierarchy. Finally, because shifts or employees that score equally on all criteria in $\mathcal{S}$ or $\mathcal{E}$ are sorted randomly, different runs with the same $\mathcal{S}$ and $\mathcal{E}$ can have different outcomes. Therefore, for each instance, we consider the averaged result over multiple runs, to account for this randomness.

After applying the described procedure on both construction methods, the method that leads to the best performance for most instances is chosen as the fixed construction method for following evaluations.

6.3. Overall frameworks

6.3.1. Selection of neighborhoods

Given the fixed construction method, the next step is to determine which neighborhood structures should be used in the overall frameworks to obtain the best performance. Again, we cannot test all possible combinations of neighborhood structures, especially since multiple vertical and horizontal swapping and change neighborhood structures with different block lengths can be applied. To select the neighborhoods, we apply a similar method as in Section 6.2: the candidate neighborhoods, as described in Section 5.2, are placed in an ordered set $C_{\mathcal{N}}$. We let $\mathcal{N}$ denote the set of neighborhoods to be used in the overall frameworks shown in Algorithms 3-6. $\mathcal{N}$ is initialized as an empty set, and since the overall frameworks cannot function with an empty set $\mathcal{N}$, this means that the baseline for this method is the performance of the previously fixed construction method. Neighborhoods from $C_{\mathcal{N}}$ are then added iteratively to $\mathcal{N}$, based on their effect on the performance of the overall framework. For this, for each neighborhood in $C_{\mathcal{N}}$ separately, we test the performance of the overall framework after adding only this neighborhood to $\mathcal{N}$. The neighborhood that results in the best performance, in terms of the objective function value, over most of the instances is then permanently added to $\mathcal{N}$. After adding this best neighborhood, the addition of the other neighborhoods in $C_{\mathcal{N}}$ is tested in the same way, after which again the best criterion is added to $\mathcal{N}$. This process is repeated until either $\mathcal{N}$ is equal to $C_{\mathcal{N}}$ or the addition of none of the remaining neighborhoods leads to better performance than already obtained in the previous iteration (or the initial baseline, in case of the first iteration). Pseudocode of this method is shown in Algorithm 8. To reduce the number of possible combinations of neighborhoods, Algorithm 8 is only applied with a block length of 1 for the horizontal swapping, vertical swapping and change neighborhood structures.

Algorithm 7 Determine shift and employee sorting criteria for construction methods

```
Input: sets of candidate shift and employee sorting criteria, $C_\mathcal{S}$ and $C_\mathcal{E}$, test instances $J_1, ..., J_M$, and objective
       function $f : X \to \mathbb{R}$ for solution space $X$
 1: Initialize ordered sets of shift and employee sorting criteria to be used in the construction method, $\mathcal{S}$ and $\mathcal{E}$,
    as empty sets
 2: Set the number of runs per instance to $K$
 3: Apply the construction method with sorting criteria sets $\mathcal{S}$ and $\mathcal{E}$ $K$ times on each instance $J_1, ..., J_M$,
    resulting in solutions $x_1^1, ..., x_1^k, x_2^1, ...x_2^k, ..., x_M^1, ..., x_M^k \in X$
 4: Save the average objective function values over the $K$ runs for all $M$ instances as
    $\mathbf{y}^0 = [\sum_{k=1}^K f(x_1^k)/K, ..., \sum_{k=1}^K f(x_M^k)/K]$
 5: Set $noMorePerformanceGainedByAdditionalCriterion = false$
 6: while ($C_\mathcal{S} \neq \emptyset$ OR $C_\mathcal{E} \neq \emptyset$) AND $noMorePerformanceGainedByAdditionalCriterion = false$ do
 7:     for each shift sorting criterion $c_\mathcal{S}$ in $C_\mathcal{S}$ do
 8:         Set $\mathcal{S}' = \mathcal{S} \cup \{c_\mathcal{S}\}$, with $c_\mathcal{S}$ last in the ordering of $\mathcal{S}'$.
 9:         Apply the construction method with sorting criteria sets $\mathcal{S}'$ and $\mathcal{E}$ $K$ times on each
            instance $J_1, ..., J_M$, resulting in solutions $x_1^1, ..., x_1^k, x_2^1, ...x_2^k, ..., x_M^1, ..., x_M^k \in X$
10:         Save the average objective function values over the $K$ runs for all $M$ instances as
            $\mathbf{y}^{c_\mathcal{S}} = [\sum_{k=1}^K f(x_1^k)/K, ..., \sum_{k=1}^K f(x_M^k)/K]$
11:     end for
12:     for each employee sorting criterion $c_\mathcal{E}$ in $C_\mathcal{E}$ do
13:         Set $\mathcal{E}' = \mathcal{E} \cup \{c_\mathcal{E}\}$, with $c_\mathcal{E}$ last in the ordering of $\mathcal{E}'$.
14:         Apply the construction method with sorting criteria sets $\mathcal{S}$ and $\mathcal{E}'$ $K$ times on each
            instance $J_1, ..., J_M$, resulting in solutions $x_1^1, ..., x_1^k, x_2^1, ...x_2^k, ..., x_M^1, ..., x_M^k \in X$
15:         Save the average objective function values over the $K$ runs for all $M$ instances as
            $\mathbf{y}^{c_\mathcal{E}} = [\sum_{k=1}^K f(x_1^k)/K, ..., \sum_{k=1}^K f(x_M^k)/K]$
16:     end for
17:     Let $Z = \{0\} \cup C_\mathcal{S} \cup C_\mathcal{E}$
18:     for each $z \in Z$ do
19:         Create binary array $\mathbf{u}^z = [u_1^z, ..., u_M^z]$, where $u_j^z$ equals 1, if $y_j^z = \min_{z \in Z} y_j^z$, and 0 otherwise,
            for each $j = 1, ..., M$
20:     end for
21:     Let $z^* = \text{argmax}_{z \in Z} \sum_{j=1}^M u_j^z$
22:     if multiple elements $z \in Z$ attain $\max_{z \in Z} \sum_{j=1}^M u_j^z$ then
23:         Let $z^* = \text{argmin}_{z \in Z} \sum_{j=1}^M y_j^z / M$
24:         if multiple elements $z \in Z$ attain $\min_{z \in Z} \sum_{j=1}^M y_j^z / M$ then
25:             Choose $z^*$ randomly among elements $z \in Z$ that attain $\min_{z \in Z} \sum_{j=1}^M y_j^z / M$
26:         end if
27:     end if
28:     if $z^* \in C_\mathcal{S}$ then
29:         Set $\mathcal{S} = \mathcal{S} \cup \{z^*\}$, with $z^*$ last in the ordering of $\mathcal{S}$
30:         Set $C_\mathcal{S} = C_\mathcal{S} \backslash \{z^*\}$
31:         Set $\mathbf{y}^0 = \mathbf{y}^{z^*}$
32:     else if $z^* \in C_\mathcal{E}$ then
33:         Set $\mathcal{E} = \mathcal{E} \cup \{z^*\}$, with $z^*$ last in the ordering of $\mathcal{E}$
34:         Set $C_\mathcal{E} = C_\mathcal{E} \backslash \{z^*\}$
35:         Set $\mathbf{y}^0 = \mathbf{y}^{z^*}$
36:     else
37:         $noMorePerformanceGainedByAdditionalCriterion = true$
38:     end if
39: end while
Output: resulting sets $\mathcal{S}$ and $\mathcal{E}$
```

In order to test the use of different neighborhoods in Variable Neighborhood Search, we need to place the applied neighborhoods in some order. During the execution of Algorithm 8, we fix the ordering of the neighborhoods in $C_{\mathcal{N}}$, based on the ordering of neighborhoods of Abdelghany et al. (2021b) for Variable Neighborhood Search, i.e. first the coverage-focused neighborhoods, followed by the horizontal and vertical swapping neighborhoods. The change neighborhood, which Abdelghany et al. (2021b) did not apply, is placed at the end, as it contains all possible alternative shift assignments for each employee on every day. The used order is shown in Table 6.3. Additionally, the set of perturbation neighborhoods has to be defined. For this, we apply a setup similar to Abdelghany et al. (2021b) and Goh et al. (2022): we let the set of perturbation neighborhoods consist of both horizontal and vertical swapping neighborhoods with block lengths of 2 up to 6, as well as the change neighborhood. This way, the perturbation mostly swaps around the existing shifts in the roster, and sometimes it adds or removes a shift or changes it to another shift type. A perturbation limit of 5% is used, such that perturbation operations are applied until the roster penalty is changed by at least 5%.

Table 6.3: Fixed order of neighborhood structures used for Variable Neighborhood Search in Algorithm 8

Rank	**Neighborhood structure**
1	UC1
2	UC2
3	UC3
4	H1
5	V1
6	C1

Similarly, to test Simulated Annealing, we need to set a probability for each applied neighborhood to be sampled in every iteration (line 8 in Algorithm 6). Because we do not have a basis on which to determine these probabilities a priori, we fix them to be equal for all applied neighborhoods during the execution of Algorithm 8. Note that the ordering of the neighborhood structures in $C_{\mathcal{N}}$ and $\mathcal{N}$ is irrelevant for Simulated Annealing. Finally, the algorithm parameters for Simulated Annealing, described in Section 5.3.2, have to be fixed to some value. Firstly, we set M_s to 100 and M_a to ρM_s with $\rho = 0.2$, similar to Ceschia et al. (2023). The initial and minimal temperatures were chosen by trial and error with different orders of magnitudes, until values were found for which the penalty did not initially increase to values as high as multiples of the penalty after construction, and finally roughly converged to a single value. An additional condition for the initial temperature was that even a penalty increase of 500, i.e. the equivalent of one unassigned priority shift, could still be accepted with a probability that is not negligible. The values $T^{start} = 50$ and $T^{min} = 0.1$ satisfied these conditions, and thus, were fixed. During this process, it was noted that the overall performance did not vary greatly with different values for T^{start} and T^{min}, within reasonable orders of magnitude. An expected number of iterations I was determined by some initial tests, where the number of iterations that could be executed within 10 minutes was counted for all nine instances. The smallest number, thus coming from the instance where an iteration requires the most computation time on average, was roughly 15 million, so I was set to 15,000,000. The cooling rate α is calculated from the other parameters as follows: $\alpha = \left(T^{min}/T^{start}\right)^{M_s/I}$. Note that the time limit of 10 minutes is not reached for instances where the average computation time of a single iteration is smaller, and in cases where the limit for accepted moves before cooling, M_a, is frequently reached earlier than the limit for sampled moves, M_s.

Block lengths After applying Algorithm 8, it is tested whether adding neighborhood structures with greater block lengths improves performance. This is done by iteratively adding horizontal swapping, vertical swapping and change neighborhood structures of increasing block lengths, until adding neighborhood structures of a greater block length does not improve the performance of the overall framework anymore. In Variable Neighborhood Search, the neighborhood structures with greater block lengths are added directly after the same type of neighborhood structure with preceding block length. For example, given the ordering shown in Table 6.3, H2 is added after H1 and before V1.

6.3.2. VNS: neighborhood orderings

For Variable Neighborhood Search, also the order in which the neighborhoods are placed is relevant for its performance. Although some logic can be applied to devising the ordering, based on for example the sizes of the neighborhoods and the types of corresponding operations, this is not guaranteed to lead to the best performing

Algorithm 8 Method to select neighborhood structures to use in one of the overall frameworks

```
Input: ordered set of candidate neighborhood structures $C_{\mathcal{N}}$, instances $J_1, ..., J_M$, fixed construction method
       and its averaged results on the instances $\mathbf{y}^0 = [y_1^0, ..., y_M^0]$, obtained in the execution of Algorithm 7,
       where $y_j^0$ is the averaged result of the fixed construction method on instance $J_j$,
       and objective function $f : X \to \mathbb{R}$ for solution space $X$
1: Initialize ordered set of neighborhoods $\mathcal{N}$, to be used in the overall framework, as an empty set
2: Set the number of runs per instance to $K$
3: Set $noMorePerformanceGainedByAdditionalNeighborhood = false$
4: while $\mathcal{N} \neq C_{\mathcal{N}}$ AND $noMorePerformanceGainedByAdditionalNeighborhood = false$ do
5:     for each neighborhood structure $c_{\mathcal{N}}$ in $C_{\mathcal{N}}$ AND not in $\mathcal{N}$ do
6:         Set $\mathcal{N}' = \mathcal{N} \cup \{c_{\mathcal{N}}\}$, following the same ordering as $C_{\mathcal{N}}$
7:         Apply the fixed construction method and the overall framework with set of neighborhood
           structures $\mathcal{N}'$ $K$ times on each instance $J_1, ..., J_M$, resulting in solutions
           $x_1^1, ..., x_1^K, x_2^1, ...x_2^K, ..., x_M^1, ..., x_M^K \in X$
8:         Save the average objective function values over the $K$ runs for all $M$ instances as
           $\mathbf{y}^{c_{\mathcal{N}}} = [\sum_{k=1}^{K} f(x_1^k)/K, ..., \sum_{k=1}^{K} f(x_M^k)/K]$
9:     end for
10:    Let $Z = \{0\} \cup (C_{\mathcal{N}} \backslash \mathcal{N})$
11:    for each $z \in Z$ do
12:        Create binary array $\mathbf{u}^z = [u_1^z, ..., u_M^z]$, where $u_j^z$ equals 1, if $y_j^z = \min_{z \in Z} y_j^z$, and 0 otherwise,
           for each $j = 1, ..., M$
13:    end for
14:    Let $z^* = \text{argmax}_{z \in Z} \sum_{j=1}^{M} u_j^z$
15:    if multiple elements $z \in Z$ attain $\max_{z \in Z} \sum_{j=1}^{M} u_j^z$ then
16:        Let $z^* = \text{argmin}_{z \in Z} \sum_{j=1}^{M} y_j^z/M$
17:        if multiple elements $z \in Z$ attain $\min_{z \in Z} \sum_{j=1}^{M} y_j^z/M$ then
18:            Choose $z^*$ randomly among elements $z \in Z$ that attain $\min_{z \in Z} \sum_{j=1}^{M} y_j^z/M$
19:        end if
20:    end if
21:    if $z^* \in (C_{\mathcal{N}} \backslash \mathcal{N})$ then
22:        Set $\mathcal{N} = \mathcal{N} \cup \{z^*\}$, following the same ordering as $C_{\mathcal{N}}$
23:        Set $\mathbf{y}^0 = \mathbf{y}^{z^*}$
24:    else
25:        $noMorePerformanceGainedByAdditionalNeighborhood = true$
26:    end if
27: end while
Output: resulting set $\mathcal{N}$
```

ordering. Therefore, several different orderings are tested, varying from the ordering described in the previous section. Firstly, should neighborhood structures with block lengths greater than one be used, it is tested to place all neighborhood structures in order of increasing block length, instead of sorting by block lengths separately for horizontal swapping, vertical swapping and change neighborhood structures. For example, instead of the ordering {H1, H2, V1, V2}, we would test the ordering {H1, V1, H2, V2}. Secondly, recall that during the execution of Algorithm 8, we use a fixed predetermined ordering of the neighborhoods, as shown in Table 6.3. Alternatively, we now test whether the order in which the neighborhood structures were added during the execution of Algorithm 8, as shown in Table 7.3, results in better performance. Finally, should any undercoverage-focused neighborhood structure be part of the outcome of Algorithm 8, it is tested to place it at the end of the ordering. This might reduce the number of futile attempts to assign additional undercovered shifts, if only very few additional undercovered shifts can be assigned in practice, leaving more computational time for other types of neighborhood operations.

6.3.3. SA: neighborhood probabilities

The probabilities with which the neighborhoods are chosen in each Simulated Annealing iteration influence the performance of the algorithm as a whole, and applying equal probabilities for all neighborhoods does not necessarily lead to the best performance. Ideally, the probabilities are adjusted such that each neighborhood is sampled frequently enough to add value and not so frequent that it results in many unproductive roster operations. In this context, a productive roster operation is a feasible operation that either improves the solution or causes a potentially useful perturbation. However, a priori it is not clear which neighborhoods should receive a higher, and which a lower probability. Additionally, since the number of possible sets of probabilities is very large, even when only large variations in individual probabilities are considered, it is not possible to test all variations in reasonable time.

Therefore, we test several different sets of neighborhood probabilities, based on observations made in the case of equal probabilities. In particular, three cases are tested: one where the probabilities are based on the number of accepted operations for each neighborhood, another based on the number of penalty improving operations, and a third based on the sum of improvements made to the roster penalty over all improving operations. Note that we consider an operation as penalty improving if it results in a lower penalty compared to the current solution. To obtain the probabilities from the accepted operations statistic, the number of accepted operations for each neighborhood is divided by the total number of accepted operations. The resulting ratios are then averaged over all nine instances, with 10 runs per instance, to obtain a single ratio for each neighborhood that can function as its probability. The probabilities based on the number of penalty improving operations and based on the sum of penalty improvements are obtained in the same way. Here, the sum of penalty improvements is the sum of the improvements to the roster penalty that was made by improving moves.

Specifically, suppose we first tested Simulated Annealing with equal probabilities for all N neighborhood structures on M instances, with K runs per instance. Let $I_{m,k}$ denote the number of Simulated Annealing iterations that were executed in run k on instance m. Furthermore, let $A_{m,k,i}$ be 1 if the considered move in iteration i in run k on instance m was accepted, and 0 otherwise. Similarly, let $B_{m,k,i}$ be 1 if the considered move in iteration i in run k on instance m was accepted *and* improved the current roster penalty, and 0 otherwise. Let $C_{m,k,i}$ be the amount by which the roster penalty was improved in iteration i in run k on instance m. Note that $C_{m,k,i}$ can be negative, but it is always positive for iterations for which $B_{m,k,i}$ is equal to 1. Finally, let $D_{m,k,i,n}$ be 1 if iteration i in run k on instance m was executed by neighborhood structure n, and 0 otherwise. Then, for the cases of accepted moves (AM), improving moves (IM) and improved penalty (IP), the probabilities for neighborhood structure n are calculated as follows:

Probability by accepted moves:
$$P_n^{AM} = \frac{\sum_{m=1}^{M} \sum_{k=1}^{K} \frac{\sum_{i=1}^{I_{m,k}} A_{m,k,i} D_{m,k,i,n}}{\sum_{i=1}^{I_{m,k}} A_{m,k,i}}}{KM} \tag{6.1}$$

Probability by improving moves:
$$P_n^{IM} = \frac{\sum_{m=1}^{M} \sum_{k=1}^{K} \frac{\sum_{i=1}^{I_{m,k}} B_{m,k,i} D_{m,k,i,n}}{\sum_{i=1}^{I_{m,k}} B_{m,k,i}}}{KM} \tag{6.2}$$

Probability by improved penalty:
$$P_n^{IP} = \frac{\sum_{m=1}^{M} \sum_{k=1}^{K} \frac{\sum_{i=1}^{I_{m,k}} B_{m,k,i} C_{m,k,i} D_{m,k,i,n}}{\sum_{i=1}^{I_{m,k}} B_{m,k,i} C_{m,k,i}}}{KM} \tag{6.3}$$

6.3.4. Dependency on construction method

The role of the construction methods is to provide a reasonably good initial solution for the following local search algorithm to improve. To test how essential the quality of the initial solution is, both local search algorithms are tested with both construction methods. Additionally, the local search algorithms are tested without a preceding construction method, i.e. with an empty roster as initial solution.

6.4. Obtaining lower bounds

Because we do not employ an exact method to solve the complete model, the optimal solutions to the problem instances are unknown. To gain some insight in the absolute quality of solutions obtained by our algorithms, we solve a simplified model with an exact solver, resulting in lower bounds for the problem instances.

The simplified model contains those hard and soft constraints that are not related to weekly rest or sequences of consecutive shifts or days off. An attempt was made to formulate the complete model as a mixed-integer program. However, especially due to the multiple possible ways to satisfy the weekly rest constraint, and the used definition of consecutive shifts or days off, as described in Chapter 4, it was expected that a very large number of additional constraints and auxiliary variables, and a considerable amount of time would be required to formulate these constraints. This would also make it less likely that a commercial solver could solve the problem instances within a reasonable amount of time and memory. Therefore, only a simplified model was solved exactly. An overview of the constraints in the original and simplified problem is given in Table 6.4. Descriptions of the hard and soft constraints can be found in Chapter 4. The mixed-integer program formulation of the simplified model is shown in Equations (6.5)-(6.32). The corresponding notation is clarified in Tables C.1 and C.2. The program is solved using the Gurobi commercial solver. Note that the coverage spread and overtime spread penalties are the only non-linear elements in the formulation.

Simplified problem formulation The rostering period is defined by the set of employees, E, the set of days, D, and the set of shift types, S. The binary decision variables are denoted by $x_{e,d,s}$: $x_{e,d,s}$ equals 1, if employee $e \in E$ is assigned to a shift of type $s \in S$ on day $d \in D$, and 0 otherwise. We denote a day-off assignment by o and let $x_{e,d,o}$ be equal to 1 if employee e is assigned to a day off on day d, and 0 otherwise. The objective function is shown in Equation (6.5): to minimize the sum of the soft constraint penalties, which are further described below.

Equation (6.6) is the expression for the coverage penalty, as described in Section 4.3. Constraints (6.7) and (6.8) force auxiliary variable $y_{d,s}$, which denotes the coverage shortage for shift type $s \in S$ on day $d \in D$, to be equal to the maximum of zero (Constraints (6.8)) and the difference between the numbers of required and assigned employees for shift type s on day d (Constraints (6.7)).

The coverage spread penalty is expressed in Equation (6.9). In Section 4.3, the calculation of this penalty was shown, based on the coverage shortages y_d for all days $d \in D$. Here, we rewrite the penalty as a quadratic and a linear term of continuous auxiliary variable z_d, where z_d represents the number of unplanned shifts on day $d \in D$ minus one, see Equation (6.4). Constraints (6.10) and (6.11) force z_d to be equal to the maximum of zero (Constraints (6.11)) and the sum of the unplanned shifts of all shift types on day d minus one (Constraints (6.10)).

$$\begin{aligned}\sum_{d\in D}\sum_{m=2}^{y_d} 100(m-1) &= \sum_{d\in D} 100\left(1+2+...+(y_d-2)+(y_d-1)\right) = \sum_{d\in D} 100\left(1+2+...+(z_d-1)+z_d\right)\\ &= \sum_{d\in D} 100\left(\frac{1}{2}\left(z_d^2+z_d\right)\right) = \sum_{d\in D} 50\left(z_d^2+z_d\right). \qquad (6.4)\end{aligned}$$

Equation (6.12) expresses the overtime hours spread penalty, as described in Section 4.3. Constraints (6.13) and (6.14) force continuous auxiliary variable u_e, which denotes the worked overtime of employee $e \in E$, to be equal to the maximum of zero (Constraints (6.14)) and the difference between the total duration of all shifts worked by employee e and its contractual working hours, specified by input parameter c_e^{con} (Constraints (6.13)). Here, the duration of shifts of type $s \in S$ in hours is given by continuous input parameter f_s for all shift types, and the shifts worked by employee e are specified by the decision variables $x_{e,d,s}$ over all days $d \in D$ and shift types $s \in S$.

The requests penalties are expressed in Equation (6.15), as described in Section 4.3: for each employee $e \in E$, a penalty is given for each time that employee e requested to work on day $d \in D$, but is assigned to a day off, requested a day off on day d, but is not assigned to a day off, requested to work shift type $s \in S$ on day d, but is not assigned to it, or requested not to work shift type s on day d, but is assigned to it. The weight of the penalty is specified by input parameter j_e.

Constraints (6.16) ensure that each employee $e \in E$ can be assigned to at most one shift per day $d \in D$. Note that we sum over all $s \in S^+$, where S^+ includes all shift types $s \in S$, as well as the day-off assignment o. Thus, by this constraint, for each employee e and day d, either $x_{e,d,s}$ equals 1 for some $s \in S$, i.e. employee e is assigned to shift type s on day d, or $x_{e,d,o}$ equals 1, i.e. employee e is assigned to a day off on day d.

The required skills constraint is shown in Constraints (6.17). Binary input parameter $k_{e,s}$ equals 1 if employee $e \in E$ has the required skills to be assigned to shifts of type $s \in S$, and 0 otherwise. We can only have that employee e is assigned to a shift of shift type s on any day $d \in D$, i.e. $x_{e,d,s}$ equals 1, if employee e has the required skills for it, i.e. $k_{e,s}$ equals 1.

Constraints (6.18) express the maximum workload constraint. The duration of shifts of type $s \in S$ in hours is given by continuous input parameter f_s for all shift types. Furthermore, the maximum workload of employee $e \in E$ in hours is given by continuous input parameter c_e^{max}. The total duration of all worked shifts by employee e, which are specified by binary decision variables $x_{e,d,s}$, must be smaller than c_e^{max}, for each employee $e \in E$.

The fixed assignments constraints are shown in Constraints (6.19) and (6.20). Constraints (6.19) ensure that the shift assignment of employee $e \in E$ on day $d \in D$ complies with any predetermined fixed assignment of type $s \in S^+$ for that employee on that day. The fixed assignments are specified by binary input parameter $m_{e,d,s}$, which is 1 if employee e has a fixed assignment of shift type s on day d, and 0 otherwise. Note that if an employee is assigned to a night shift starting on day d, that shift will end somewhere during day $d+1$. Thus, to fully comply with fixed day-off assignments, Constraints (6.20) ensure that an employee is not assigned to a night shift on day d, if it has a fixed day-off assignment on day $d+1$, i.e. $m_{e,d+1,o}$ equals 1. The binary input parameter β_s is equal to 1 if shift type s is a night shift, and 0 otherwise.

In Constraints (6.21), the daily rest constraint is expressed: an employee $e \in E$ can only work both shift type $s \in S$ on day $d \in D$ and shift type t on day $d+1$, i.e. both $x_{e,d,s}$ equals 1 and $x_{e,d+1,t}$ equals 1, if either there is at least 11 hours of rest in between, or there is at least 8 hours of rest in between and the daily rest exception is used on day d for employee e. Binary input parameter $n_{s,t}$ equals 1 if there is at least 11 hours of rest in between shift type s on day d and shift type t on day $d+1$, and 0 otherwise. Similarly, binary input parameter $p_{s,t}$ equals 1 if there is at least 8 hours of rest in between, and 0 otherwise. Binary auxiliary variable $v_{e,d}$ equals 1 if the daily rest exception, which allows a rest period of between 8 and 11 hours once every 7 days, is used by employee e on day d, and 0 otherwise. Constraints (6.22) ensure that the daily rest exception is used at most once every 7 days.

The worked Sundays constraint is shown in Constraints (6.23): for each employee $e \in E$, the number of worked Sundays must be at most the maximum allowed number for that employee, which is given by input parameter r_e. Constraints (6.24) ensure that binary auxiliary variable $w_{e,d}$ equals 1 if employee $e \in E$ works any shift $s \in S$ on day $d \in D$ and day d is a Sunday, i.e. $x_{e,d,s}$ equals 1 and q_d equals 1. Constraints (6.25) ensure that $w_{e,d}$ also equals 1 if employee e works a shift of type s on day $d-1$, shift type s is a night shift, and d is a Sunday, i.e. $x_{e,d-1,s}$ equals 1, β_s equals 1 and q_d equals 1.

Constraints (6.26) show the worked night shifts constraint: for each employee $e \in E$, the total number of worked night shifts, i.e. the number of cases where $x_{e,d,s}$ equals 1 and β_s equals 1 for any shift type $s \in S$ and day $d \in D$, must be lower than the maximum allowed number of night shifts for that employee $e \in E$, which is denoted by input parameter γ_e.

The domains for the decision and auxiliary variables are shown in Constraints (6.27)-(6.32).

Objective function:

$$\min \quad (6.6) + (6.9) + (6.12) + (6.15) \tag{6.5}$$

Coverage penalty:

$$\sum_{d \in D} \sum_{s \in S} y_{d,s} \left(500 b_s + 100(1 - b_s)\right) \tag{6.6}$$

$$y_{d,s} \geq a_{d,s} - \sum_{e \in E} x_{e,d,s} \qquad \forall d \in D, s \in S \tag{6.7}$$

$$y_{d,s} \geq 0 \qquad \forall d \in D, s \in S \tag{6.8}$$

Coverage spread penalty:

$$\sum_{d \in D} 50(z_d^2 + z_d) \tag{6.9}$$

$$z_d \geq \left(\sum_{s \in S} y_{d,s} \right) - 1 \qquad \forall d \in D \tag{6.10}$$

$$z_d \geq 0 \qquad \forall d \in D \tag{6.11}$$

Overtime hours spread penalty:

$$\sum_{e \in E} u_e^2 \tag{6.12}$$

$$u_e \geq \left(\sum_{d \in D} \sum_{s \in S} x_{e,d,s} f_s \right) - c_e^{con} \qquad \forall e \in E \tag{6.13}$$

$$u_e \geq 0 \qquad \forall e \in E \tag{6.14}$$

Requests penalties:

$$\sum_{e \in E} \sum_{d \in D} j_e \left(x_{e,d,o} g_{e,d}^{on} + (1 - x_{e,d,o}) g_{e,d}^{off} \right) + \sum_{e \in E} \sum_{d \in D} \sum_{s \in S} j_e \left((1 - x_{e,d,s}) h_{e,d,s}^{on} + x_{e,d,s} h_{e,d,s}^{off} \right) \tag{6.15}$$

Single shift per day constraint:

$$\sum_{s \in S^+} x_{e,d,s} = 1 \qquad \forall e \in E, d \in D \tag{6.16}$$

Required skills constraint:

$$x_{e,d,s} \leq k_{e,s} \qquad \forall e \in E, d \in D, s \in S \tag{6.17}$$

Maximum workload constraint

$$\sum_{d \in D} \sum_{s \in S} x_{e,d,s} f_s \leq c_e^{max} \qquad \forall e \in E \tag{6.18}$$

Fixed assignments constraint:

$$x_{e,d,s} \geq m_{e,d,s} \qquad \forall e \in E, d \in D, s \in S^+ \tag{6.19}$$

$$x_{e,d,s} \beta_s \leq 1 - m_{e,d+1,o} \qquad \forall e \in E, d \in D : d < |D|, s \in S \tag{6.20}$$

Daily rest constraint:

$$x_{e,d,s} + x_{e,d+1,t} \leq 1 + n_{s,t} + v_{e,d} p_{s,t} \qquad \forall e \in E, d \in D : d < |D|, s, t \in S \tag{6.21}$$

$$\sum_{i=0}^{6} v_{e,d+i} \leq 1 \qquad \forall e \in E, d \in D : d \leq |D| - 6 \tag{6.22}$$

Worked Sundays constraint:

$$\sum_{d \in D} w_{e,d} \leq r_e \qquad \forall e \in E \tag{6.23}$$

$$w_{e,d} \geq q_d (1 - x_{e,d,o}) \qquad \forall e \in E, d \in D \tag{6.24}$$

$$w_{e,d} \geq q_d \sum_{s \in S} x_{e,d-1,s} \beta_s \qquad \forall e \in E, d \in D : d > 1 \tag{6.25}$$

Worked night shifts constraint:

$$\sum_{d \in D} \sum_{s \in S} x_{e,d,s} \beta_s \leq \gamma_e \qquad \forall e \in E \tag{6.26}$$

Variable domains:

$$x_{e,d,s} \in \{0, 1\} \qquad \forall e \in E, d \in D, s \in S^+ \tag{6.27}$$

$$y_{d,s} \in \mathbb{R} \qquad \forall d \in D, s \in S \tag{6.28}$$

$$z_d \in \mathbb{R} \qquad \forall d \in D \tag{6.29}$$

$$u_e \in \mathbb{R} \qquad \forall e \in E \tag{6.30}$$

$$v_{e,d} \in \{0, 1\} \qquad \forall e \in E, d \in D \tag{6.31}$$

$$w_{e,d} \in \{0, 1\} \qquad \forall e \in E, d \in D \tag{6.32}$$

6.5. Comparison with ORTEC's algorithms

After testing the different variations of the two construction methods and the two overall frameworks, the best working combinations are compared to the performance of the AutoRoster local search algorithm and the solver incorporated in the ORTEC for Workforce Scheduling (OWS) software product.

AutoRoster The AutoRoster algorithm is located in the same C# implementation as the methods developed in this thesis, and thus, can easily be used on our problem description. However, the AutoRoster local search algorithm allows for assigning shifts of which the shift type is not required on that day, i.e. the coverage requirement of that shift type is zero on that day. Therefore, we also test our overall frameworks such that this is allowed, for fair comparison. Table 6.4 gives an overview of the resulting problem description. The AutoRoster local search algorithm consists of a construction method and a local search method that considers horizontal swaps and shift assignment changes. AutoRoster is a solution method that is owned by Staff Roster Solutions[1], and for which ORTEC has a license to use it for research purposes.

OWS solver Since the OWS solver is incorporated in a large software product, it cannot be easily adjusted to use our problem description. To obtain a fair comparison, we omit the coverage spread and preferred shift and day-off sequence length penalties. Additionally, since the OWS solver does not allow for assigning more shifts than required by the coverage requirements, we add a hard maximum coverage constraint. The maximum shift coverage of each shift type on each day is set to be equal to the corresponding desired shift coverage, as specified in the shift coverage soft constraint. An overview of the resulting problem description is given in Table 6.4. Also, because of differences in how the coverage penalty is handled, both in terms of shift types of different priorities, and compared to the other penalties, all shifts are given the same priority and a weight of 10,000. Finally, all day and shift requests receive the same penalty of 100, and the numbers of overtime hours are rounded up before calculating the overtime hours spread penalty. Because the process of aligning the data instances used by Gurobi, Simulated Annealing and AutoRoster with the data used by OWS is elaborate and time-consuming, only the trauma instances are prepared, to obtain a first illustration of mutual differences in performance. The OWS solver consists of a construction method, a population-based meta-heuristics, a local search method and a ruin-and-recreate framework.

Table 6.4: Overview of modifications to the original problem described in Chapter 4, as used for comparison with Gurobi, the AutoRoster local search algorithm and the OWS solver. *Note that disallowing non-required shifts is an implicit hard constraint, resulting from the neighborhood restrictions described in Section 5.2, and that the corresponding constraints form a subset of the maximum shift coverage constraints.

			Model used by		
Constraints		**Original model**	**Gurobi**	**AutoRoster**	**OWS**
Hard constraints	Daily rest	x	x	x	x
	Weekly rest	x		x	x
	Worked Sundays	x	x	x	x
	Rest after consecutive night shifts	x		x	x
	Consecutive shifts with night shifts	x		x	x
	Worked night shifts	x	x	x	x
	Maximum workload	x	x	x	x
	Required skills	x	x	x	x
	Maximum shift coverage				x
	Disallow non-required shifts*	x			x
Soft constraints	Shift coverage	x	x	x	x
	Shift coverage spread	x	x	x	
	Overtime hours spread	x	x	x	x
	Day-on/off and shift-on/off requests	x	x	x	x
	Preferred shift sequence length	x		x	
	Preferred day-off sequence length	x		x	

[1]https://www.staffrostersolutions.com/

7 Results

This chapter displays the results that were obtained after applying the evaluation methods described in Chapter 6. All tests were executed on the nine instances described in Section 6.1. However, in view of readability, only the figures for results on the April instances are shown in this chapter. The figures for the May and June instances can be found in Appendix D. Note that the results differ greatly between different instances, in terms of the size of the total penalties, and that differences between different methods or settings are often relatively small compared to the total penalties. Therefore, the y-axis in the presented figures does not start at zero, and the figures do not share the same scale for different instances.

7.1. Construction methods

Algorithm 7 for determining the shift and employee sorting criteria was applied to both construction methods, shown in Algorithms 1 and 2. The number of runs per instance, K, was set to 25. The resulting sets of shift and employee sorting criteria, including their corresponding hierarchies, are shown in Table 7.1.

Table 7.1: Shift and employee sorting criteria resulting from applying Algorithm 7 to both construction methods. The order in which the criteria were added to the sets is indicated by the numbers in brackets.

	Construction-per-shift		**Construction-per-employee**	
Rank	**Shift sorting criteria**	**Employee sorting criteria**	**Shift sorting criteria**	**Employee sorting criteria**
1	Coverage requirements (1)	Available working time (2)	Night shifts (2)	Available working time (1)
2	Priority shifts (5)	Day-on/off requests (3)	Coverage requirements (3)	
3		Shift-on/off requests (4)	Number of available employees (4)	

It was noted that for Construction-per-shift, the criteria of priority shifts, day requests and shift requests were added after the criteria of coverage requirements and available working time. This means that the former criteria caused a smaller performance gain than the latter ones. However, the former criteria only affect a relatively small part of the shifts or employees. Therefore, it might be the case that these criteria are, in fact, very effective for the shifts or employees they affect, but their overall impact is relatively low due to the small number of shifts or employees they affect. For this reason, an additional test was done where the order of the former and latter criteria was reversed. The resulting order is shown in Table 7.2. This alternative ordering resulted in better performance for 8 out of 9 instances, mostly due to smaller numbers of unplanned priority shifts and unsatisfied requests, and was therefore considered as the final ordering of the criteria for Construction-per-shift. Construction-per-employee did not have a similar situation in its resulting sorting criteria. Still, based on the improved performance of the alternative ordering for Construction-per-shift, additional tests were done where the priority shifts and shift requests criteria were added at the top of the hierarchy of the shift sorting criteria for Construction-per-employee. This lead to slightly improved performance for 7 out of 9 instances, so the criteria shown in Table 7.2 were considered as the final criteria for Construction-per-employee.

Results of both construction methods with their final sorting criteria are shown in Figures 7.1 and D.1. Appendix D contains the results in numerical values of both the sorting criteria obtained by the execution of Algorithm 7, shown in Table 7.1, and the final sorting criteria, shown in Table 7.2, in Tables D.1 and D.2, as well as

Table 7.2: Alternative ordering of the sorting criteria for Construction-per-shift and Construction-per-employee

	Construction-per-shift		**Construction-per-employee**	
Rank	**Shift sorting criteria**	**Employee sorting criteria**	**Shift sorting criteria**	**Employee sorting criteria**
1	Priority shifts	Day-on/off requests	Shift-on/off requests	Available working time
2	Coverage requirements	Shift-on/off requests	Priority shifts	
3		Available working time	Night shifts	
4			Coverage requirements	
5			Number of available employees	

the breakdown of the total roster penalties into the different penalty categories in Tables D.3 and D.5, and the run times and the roster properties related to the different penalty categories in Tables D.4 and D.6.

It is clear that Construction-per-shift outperforms Construction-per-employee, as it significantly scores better in terms of the objective function value for all nine instances. This is also the case for almost all penalty categories individually. Only in terms of the overtime spread penalty and the preferred shift sequence length penalty, Construction-per-employee scores better on 3 and 2 out of 9 instances, respectively. With respect to the run times, both methods finish well within 1 second for all instances, and neither method holds a significant advantage over the other.

Based on the results described above, Construction-per-shift with the sorting criteria shown in Table 7.2 was fixed for the following tests with the overall frameworks.

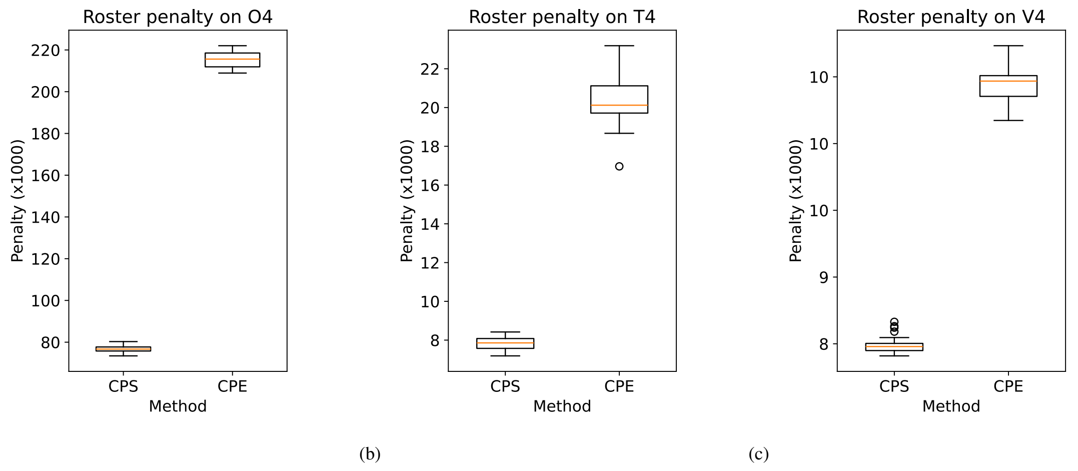


Figure 7.1: Box plots of the results of Construction-per-shift (CPS) and Construction-per-employee (CPE), using the sorting criteria shown in Table 7.2, on the April instances over 25 runs per instance.

7.2. Overall frameworks

7.2.1. Selection of neighborhoods

Algorithm 8 was applied to both Variable Neighborhood Search and Simulated Annealing. The number of runs per instance, K, was set to 10. The resulting sets of neighborhood structures are shown in Table 7.3. These sets of neighborhood structures were thus used in further tests.

Block lengths After having obtained the different neighborhood structures for Variable Neighborhood Search and Simulated Annealing, it was tested whether the addition of Cx, Hx and Vx neighborhood structures with block lengths x greater than 1 could improve performance. Note that the maximum block lengths of Cx, Hx and Vx neighborhood structures were incremented simultaneously. The results are shown in Figures 7.2, D.2 and D.3. For Simulated Annealing, a general trend can be observed where the performance increases with the addition of Cx, Hx and Vx up to a block length x of around 4, after which the performance decreases again with the addition

Table 7.3: Sets of neighborhood structures resulting from applying Algorithm 8 to Variable Neighborhood Search (VNS) and Simulated Annealing (SA), shown in the order by which they were added to the sets. The neighborhood structures and their abbreviations are introduced in Section 5.2.

	Neighborhood structures	
Rank	**VNS**	**SA**
1	UC2	C1
2	C1	H1
3	V1	V1
4	H1	
5	UC1	

of neighborhood structures with greater block lengths. Therefore, from this point forward, Simulated Annealing was applied with the Cx, Hx and Vx neighborhood structures with block lengths 1, 2, 3 and 4.

For Variable Neighborhood Search, maximum block lengths up to 20 were tested, and the general trend is that a larger maximum block length results in increased performance. Note that Hx neighborhood structures with block lengths x greater than 15 were not added, because in a scheduling period of a month there are no two non-overlapping blocks of 16 or more days that can be swapped for an employee. Because the performance appeared to stabilize at the largest tested maximum block lengths, no tests were executed with maximum block lengths greater than 20. When looking more specifically to the average performance at different maximum block lengths, a maximum block length of 19 appeared to have the best performance by a small margin compared to similar maximum block lengths. Afterwards, it was observed that Cx and Hx neighborhood structures with block lengths x larger than roughly 7 and 10, respectively, did not result in any penalty improving moves (see Table D.9). Thus, for further tests, we applied Variable Neighborhood Search with Cx, Hx and Vx neighborhood structures with block lengths 1 through 7, 1 through 10, and 1 through 19, respectively.

7.2.2. VNS: neighborhood orderings

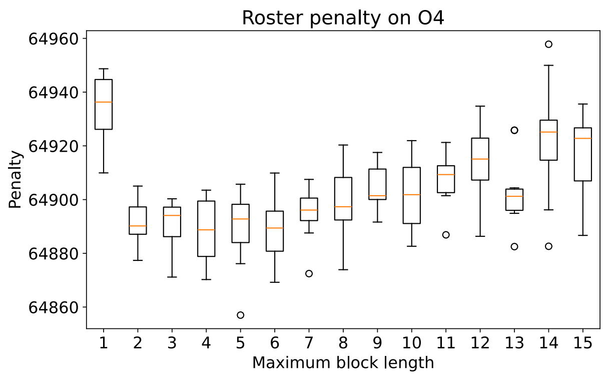
Table 7.4 shows the neighborhood orderings that were tested for Variable Neighborhood Search, as described in Section 6.3.2. The results of these orderings are shown in Figures 7.3 and D.4. It can be observed that the differences between the performance with the different orderings are relatively small. Overall, none of the orderings appears to generally yield better results than the original ordering. Especially between the original ordering (O) and the ordering with overall increasing block length (IBL), the results are very close. The distinction between these two was finally based on the averaged differences over the nine instances, where the original ordering scored slightly better (see Table D.10). Therefore, the original ordering is used in further tests.

7.2.3. SA: neighborhood probabilities

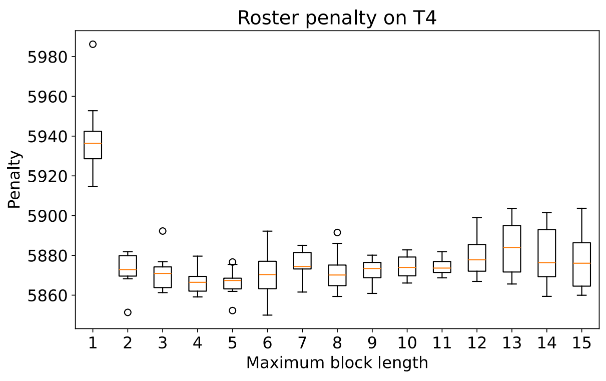
Table 7.5 shows the neighborhood probability distributions, that were calculated as described in Section 6.3.3, and tested for Simulated Annealing. General trends in these distributions are that the Vx neighborhood structures receive higher probabilities than Hx and Cx, and that the neighborhood probabilities decrease with increasing block lengths. The results of these distributions, compared to the case of equal probabilities for all neighborhoods, are shown in Figures 7.4 and D.5. Firstly, it can be observed that the differences between the performance with the different distributions are relatively small. Still, the setting based on the number of improving moves yields somewhat better results on most instances, and thus, this setting is used in further tests.

7.2.4. Dependency on construction method

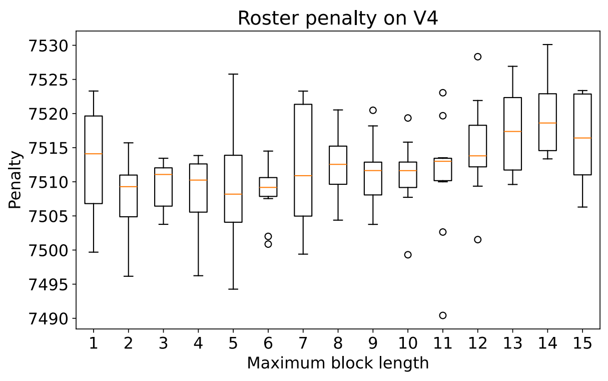
The results of the different pairs of overall frameworks and construction methods are shown in Figures 7.5 and D.6. Although Construction-per-shift results in initial solutions with much smaller penalties than those obtained with Construction-per-employee, and much more so compared to empty rosters, the performance of Simulated Annealing and Variable Neighborhood Search does not appear to be significantly affected when preceded by Construction-per-employee or no construction method at all, instead of Construction-per-shift. This indicates that the neighborhood structures used for Simulated Annealing and Variable Neighborhood Search contain the necessary types of roster operations to both construct a roster and further improve it.



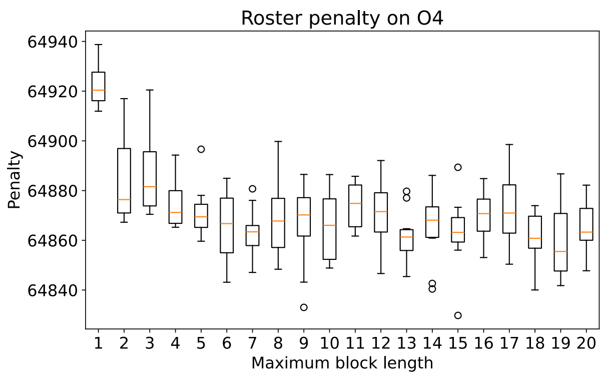
(a)



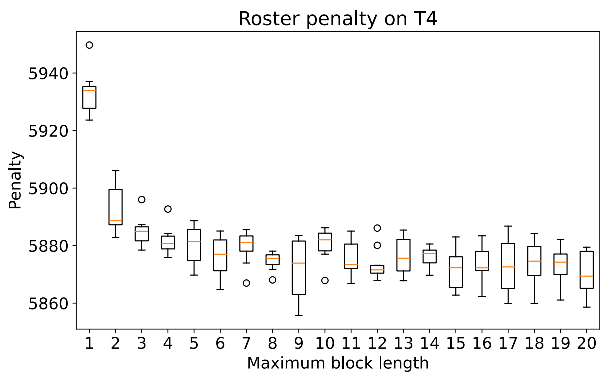
(b)



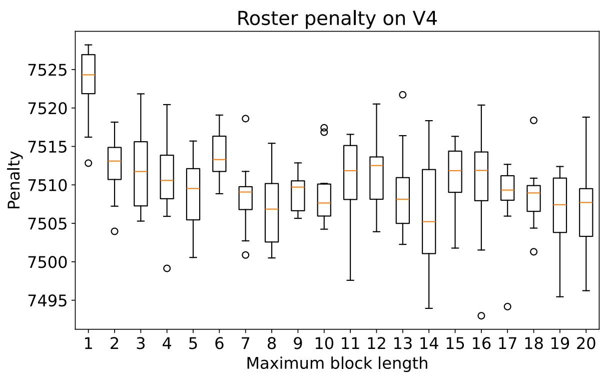
(c)



(d)



(e)



(f)

Figure 7.2: Box plots of the results of Simulated Annealing (a)-(c) and Variable Neighborhood Search (d)-(f) with different maximum block lengths on the April instances over 10 runs per instance.

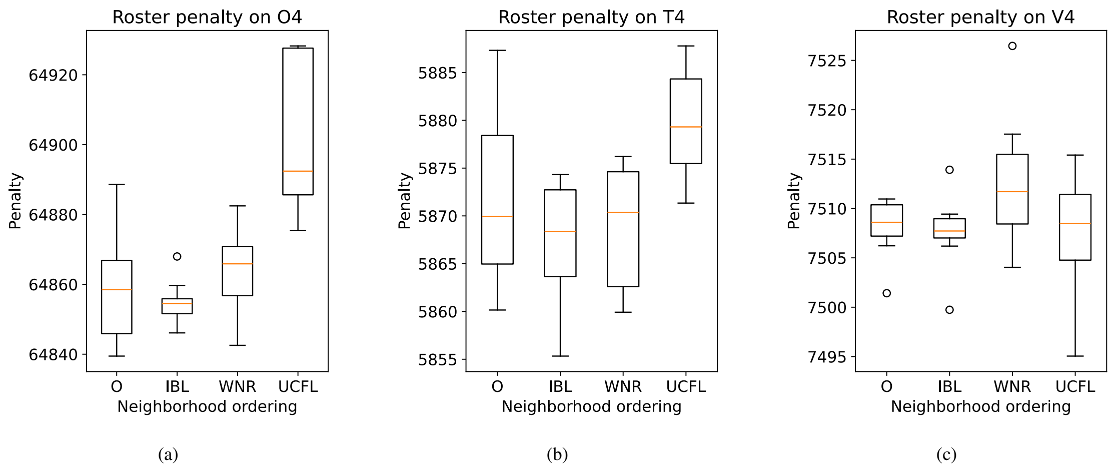


Figure 7.3: Box plots of the results of Variable Neighborhood Search with different neighborhood orderings on the April instances over 10 runs per instance. The tested orderings are displayed in Table 7.4. Two high-valued outliers for ordering UCFL on instance O4 were omitted from Subfigure (a) for clarity.

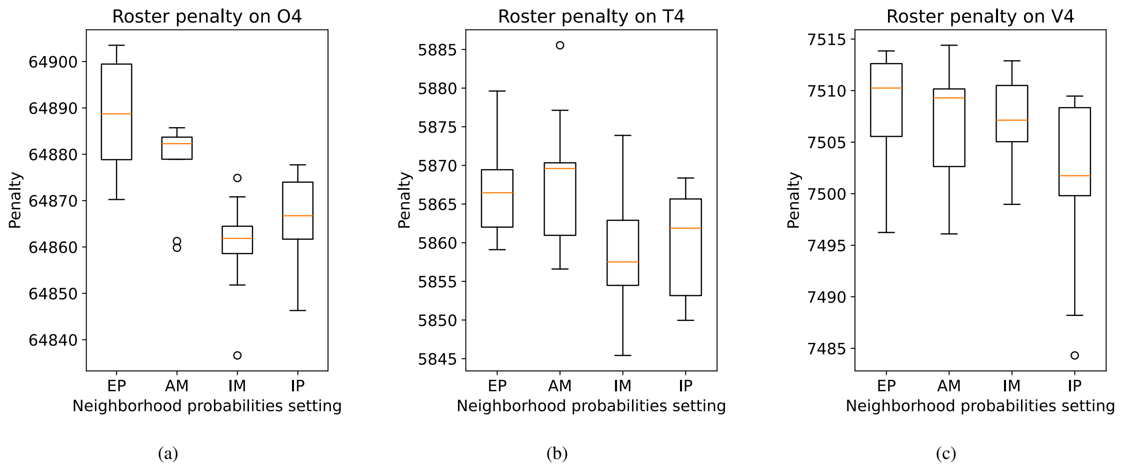
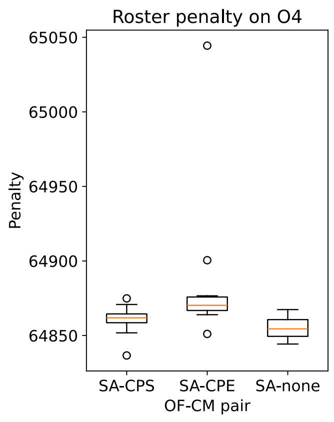
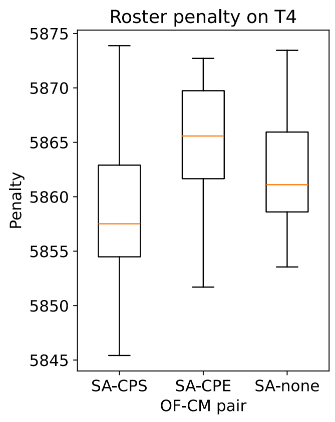


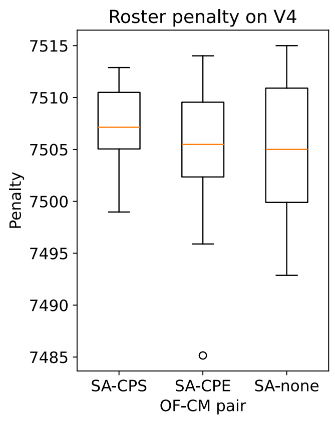
Figure 7.4: Box plots of the results of Simulated Annealing with different neighborhood probability distributions on the April instances over 10 runs per instance. The shown settings are equal probabilities (EP) and probabilities by accepted moves (AM), improving moves (IM) and improved penalty (IP), and the corresponding probabilities are shown in Table 7.5.



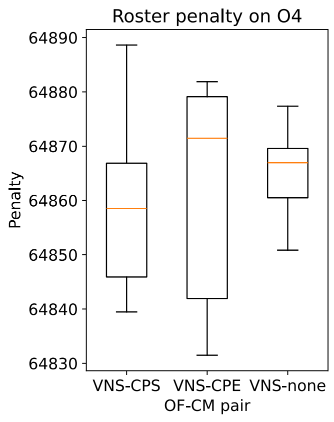
(a)



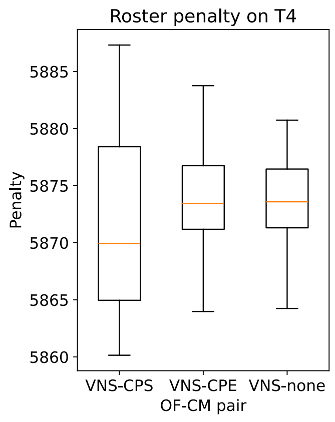
(b)



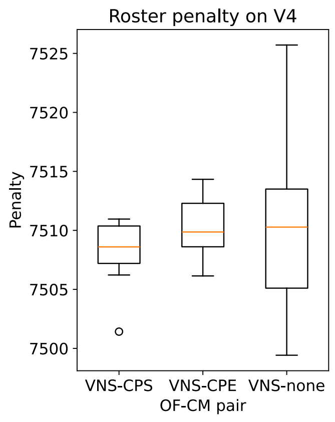
(c)



(d)



(e)



(f)

Figure 7.5: Box plots of the results of different pairs of overall frameworks (OF), i.e. Simulated Annealing (SA) and Variable Neighborhood Search (VNS), and construction methods (CM), i.e. Construction-per-shift (CPS), Construction-per-employee (CPE) and no construction (none), on the April instances over 10 runs per instance.

Table 7.4: Tested neighborhood orderings for Variable Neighborhood Search: the original ordering (O), as described in Section 6.3.1, as well as the ordering with overall increasing block length (IBL), the ordering following the results from the execution of Algorithm 8 (WNR), and the ordering with the undercoverage focused neighborhoods last (UCFL), as described in Section 6.3.2.

Neighborhood ordering			
O	**IBL**	**WNR**	**UCFL**
UC1	UC1	UC2	H1
UC2	UC2	C1	...
H1	H1	...	H10
...	V1	C7	V1
H10	C1	V1	...
V1	...	...	V19
...	H7	V19	C1
V19	V7	H1	...
C1	C7	...	C7
...	H8	H10	UC1
C7	V8	UC1	UC2
	...		
	H10		
	V10		
	V11		
	...		
	V19		

Table 7.5: Tested neighborhood probability distributions, based on the numbers of accepted moves (AM), improving moves (IM) and the improved penalty (IP) of the neighborhood structures obtained from the case where all neighborhood probabilities are equal. Also the aggregate probabilities of neighborhoods of the same operation type, i.e. Hx, Vx or Cx, and of the same block length x are shown. The case of equal probabilities (EP) is displayed for comparison.

		Probability by		
Neighborhood structure(s)	**EP**	**AM**	**IM**	**IP**
H1	0.0833	0.118199	0.154385	0.207851
H2	0.0833	0.019165	0.027736	0.051106
H3	0.0833	0.011059	0.016471	0.031208
H4	0.0833	0.006701	0.009870	0.019286
V1	0.0833	0.256920	0.157966	0.115742
V2	0.0833	0.146636	0.195767	0.136398
V3	0.0833	0.093488	0.139979	0.106827
V4	0.0833	0.070068	0.116519	0.097278
C1	0.0833	0.214583	0.133434	0.166114
C2	0.0833	0.043728	0.029915	0.038309
C3	0.0833	0.014081	0.012157	0.018924
C4	0.0833	0.005371	0.005801	0.010957
Hx	0.333	0.155124	0.208462	0.309451
Vx	0.333	0.567112	0.610231	0.45245
Cx	0.333	0.277764	0.181307	0.234304
$x = 1$	0.25	0.589702	0.445784	0.489706
$x = 2$	0.25	0.209529	0.253418	0.225813
$x = 3$	0.25	0.118629	0.168607	0.156959
$x = 4$	0.25	0.082140	0.132190	0.127522

7.2.5. Simulated Annealing vs. Variable Neighborhood Search

For the tests with Simulated Annealing to this point, the stopping criterion of the minimum temperature could be reached before the time limit of 10 minutes, as described in Section 6.3.1. For a fairer comparison with Variable Neighborhood Search, which always reaches the time limit, we did additional tests with Simulated Annealing, without the minimum temperature stopping criterion. Note that this can give at most a very small increase in performance, as Simulated Annealing will have roughly converged when the minimum temperature is reached. Additional tuning of the Simulated Annealing parameters for each instance could let the algorithm use the given time more effectively. The results of Simulated Annealing and Variable Neighborhood Search are shown in Figures 7.6 and D.8. The corresponding run times are shown in Figures 7.7 and D.9. The numerical values for Simulated Annealing without minimum temperature stopping criterion and Variable Neighborhood Search, and the resulting roster properties are shown in Tables D.11 and D.12. The breakdown of the total penalty into the different penalty categories can be found in Tables 7.7, 7.8 and 7.9 (columns SA (2) and VNS (2)). From this comparison, we conclude that Simulated Annealing generally outperforms Variable Neighborhood Search, although the differences are relatively small. Note that this also holds for Simulated Annealing with the minimum temperature stopping condition, which generally uses less time. Additionally, we observe that the standard deviations over the nine instances are relatively small for both methods: between 0.0093% and 0.14% of the mean for Simulated Annealing (0.077% on average), and between 0.011% and 0.23% for Variable Neighborhood Search (0.085% on average).

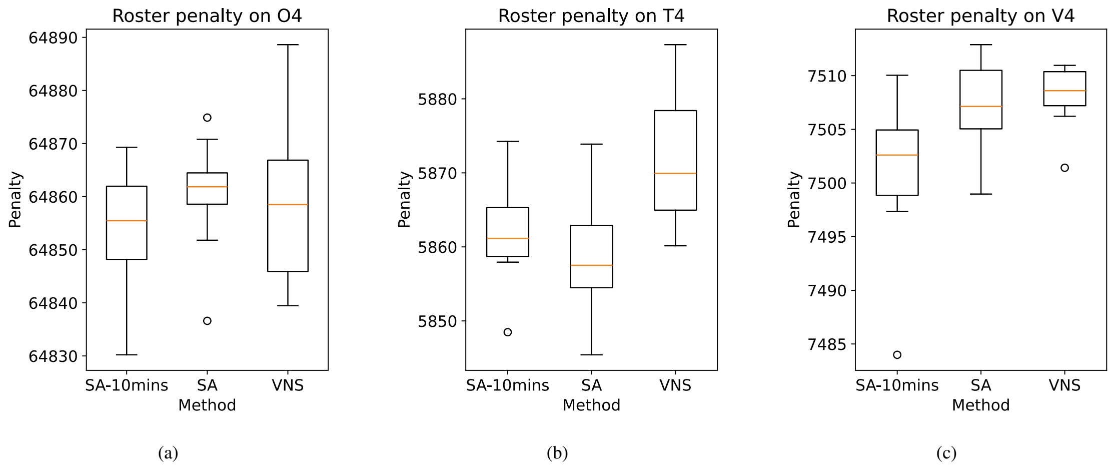


Figure 7.6: Box plots of the results of Simulated Annealing, with and without the minimum temperature stopping criterion (SA and SA-10mins, respectively), and Variable Neighborhood Search (VNS) on the April instances over 10 runs per instance. Note that SA-10mins and VNS always reach the time limit of 10 minutes, whereas SA can be stopped earlier by the minimum temperature stopping conditions, see Figures 7.7 and D.9.

7.2.6. Construction-per-shift vs. Simulated Annealing

To gain insight in the quality of an initial solution, obtained by our best performing construction method, i.e. Construction-per-shift, we compare the results of Construction-per-shift with the results of our best performing overall framework, i.e. Simulated Annealing. The results are shown in Tables 7.6, D.13 and D.14. We observe that, on all instances, the worst solution found by Simulated Annealing is still significantly better than the best solution found by Construction-per-shift. Thus, Simulated Annealing can significantly improve the constructed initial solution. Specifically, most of the improvement is gained in the coverage spread penalty and the preferred shift and day-off sequence length penalties. Note that Construction-per-shift does not take these penalties into account. In contrast, for most instances, only few additional required shifts can be assigned by Simulated Annealing compared to the initial solution constructed by Construction-per-shift, which is reflected in relatively small differences in the coverage penalty. Additionally, the deviation between the outcomes of different runs is far smaller for Simulated Annealing than for Construction-per-shift. Although it may be counterintuitive that a construction method, such as Construction-per-shift, is less stable than a probabilistic local search method, such as Simulated Annealing, this can be explained by the fact that Construction-per-shift does not take all soft constraints into account and also contains a degree of randomness in the sorting of shifts and employees that score the same on all sorting criteria.

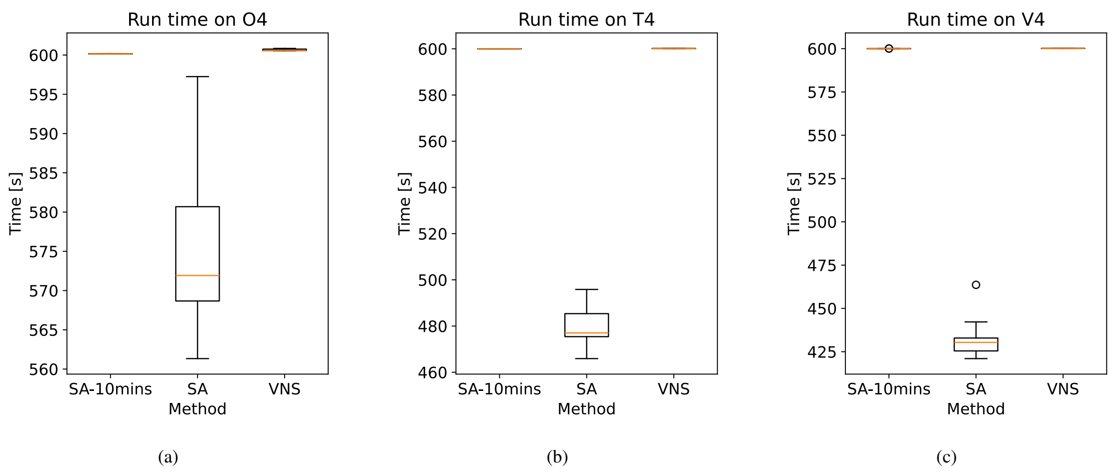


Figure 7.7: Box plots of the run times of Simulated Annealing, with and without the temperature stopping criterion (SA and SA-10mins, respectively), and Variable Neighborhood Search (VNS) on the April instances over 10 runs per instance.

Table 7.6: Results of Construction-per-shift (CPS) and Simulated Annealing (SA) on all nine instances. The mean, standard deviation (SD), minimum and maximum are shown of the total roster penalties over 25 runs for CPS and 10 runs for SA. The values are rounded somewhat for clarity.

	Construction-per-shift				**Simulated Annealing**			
Instance	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**
O4	76,812	1,823	73,442	80,287	64,854	11.3	64,830	64,869
O5	110,178	2,517	106,452	115,643	98,624	15.1	98,604	98,649
O6	93,302	2,167	89,553	98,335	82,309	7.6	82,298	82,324
T4	7,815	383	7,186	8,420	5,862	7.1	5,848	5,874
T5	8,708	299	7,979	9,266	6,647	8.8	6,630	6,661
T6	13,183	468	12,481	14,238	10,656	4.1	10,650	10,661
V4	8,493	69	8,409	8,664	7,501	7.0	7,484	7,510
V5	6,414	82	6,246	6,565	5,304	6.4	5,293	5,314
V6	19,672	755	18,209	20,922	13,987	19.8	13,946	14,013

7.3. Lower bounds

The mixed-integer program, described in Section 6.4, was solved using Gurobi 11.0. The results are shown in Tables 7.7, 7.8 and 7.9. In terms of the soft constraints, the program solved by Gurobi only contains the coverage, coverage spread, overtime hours spread and requests penalties. We refer to the sum of these four penalties as the partial objective. The remaining two soft constraints, i.e. the preferred shift sequence length and preferred day-off sequence length penalties, are referred to as the additional objective. The partial and additional objective together form the complete objective. The program solved by Gurobi contains a subset of the hard constraints and the partial objective. The corresponding results are compared to Simulated Annealing and Variable Neighborhood Search in the case with all hard constraints and the partial objective (SA (1) and VNS (1) in the tables), and in the case with all hard constraints and the complete objective (SA (2) and VNS (2) in the tables). Note that for the Gurobi, SA (1) and VNS (1) results, the additional objective was not taken into account during the execution of the algorithms, but was calculated afterwards for the resulting rosters, together with the total penalty. Because Gurobi was used on a model with fewer hard constraints, the results it obtained on the partial objective form lower bounds on the partial objective for the complete model with all hard constraints. Therefore, as the objective function value of any feasible solution forms an upper bound on the optimal objective value, any feasible solution that attains the lower bound on the partial objective is optimal with respect to the partial objective.

We find that Simulated Annealing and Variable Neighborhood Search, when applied without the additional objective (SA (1) and VNS (1) in the tables), obtain the same result as Gurobi for the partial objective in each run on 7 out of 9 instances, and are thus optimal with respect to the partial objective on those instances. Note that here only the additional objective is omitted from the original model. When we applied Simulated Annealing on the same model as Gurobi (see Table 6.4), by also omitting the hard constraints on weekly rest, rest after consecutive night shifts, and consecutive shifts with night shifts, and the non-required shifts neighborhood restrictions, it also found the optimal values on the partial objective for each run on the O6 and V6 instances. In this case, Variable Neighborhood Search found the optimal values for each run on the O6, and for some runs on the V6 instance. Therefore, it is likely that in the case of SA (1) and VNS (1), the remaining hard constraints, that were not in the Gurobi model, made it impossible to find a feasible solution that attains the lower bound on the partial objective.

Comparing the results from SA (1) to SA (2), and from VNS (1) to VNS (2), we find that the addition of the additional objective to the model only causes at most a small increase in the partial objective value, whereas it causes a significant improvement in the performance with respect to the additional objective, and thereby also with respect to the complete objective. However, from these results, we cannot draw absolute conclusions on the performance with respect to the complete objective, as we do not have a lower bound greater than zero on the additional objective.

Table 7.7: Results obtained by Gurobi (G), compared with results of Simulated Annealing (SA) and Variable Neighborhood Search (VNS), applied both on the partial objective (1) and the complete objective (2), averaged over 10 runs per instance. The partial objective contains the coverage, coverage spread, overtime hours spread and requests penalties. The complete objective contains the partial objective and the additional objective, which consists of the preferred shift sequence length and preferred day-off sequence length penalties. For Gurobi, SA (1) and VNS (1), the additional objective and the total penalty were calculated afterwards for the resulting rosters. Note that the results from Gurobi form a lower bound for the partial objective. Asterisks (*) indicate that the SA or VNS result was equal to the value found by Gurobi for each run.

	Total penalty					**Partial objective**				
		SA		**VNS**			**SA**		**VNS**	
Instance	**G**	**(1)**	**(2)**	**(1)**	**(2)**	**G**	**(1)**	**(2)**	**(1)**	**(2)**
O4	65,949.49	65,940.66	64,854.36	65,949.79	64,859.19	63,574.79	*	63,576.22	*	63,581.12
O5	100,087.30	100,079.06	98,624.09	100,095.41	98,646.05	97,308.96	*	97,311.65	*	97,318.03
O6	83,364.75	83,337.16	82,309.05	83,338.63	82,307.28	81,342.40	81,352.40	81,355.04	81,352.40	81,356.43
T4	6,549.13	6,498.46	5,862.14	6,487.54	5,871.50	4,805.07	*	4,811.67	*	4,810.57
T5	7,325.02	7,320.72	6,646.82	7,354.02	6,658.26	5,719.02	*	*	*	5,719.03
T6	11,366.86	11,365.04	10,656.46	11,347.35	10,670.41	9,785.63	*	*	*	9,785.64
V4	8,412.29	8,300.73	7,501.00	8,400.83	7,508.14	6,601.94	*	*	*	6,601.97
V5	6,309.91	6,197.50	5,303.90	6,345.45	5,313.94	4,382.31	*	4,383.61	*	4,383.57
V6	14,815.06	14,867.84	13,986.70	14,980.30	14031.96	12,680.48	12,936.67	12,958.86	12,964.82	13,000.75

Table 7.8: See the caption of Table 7.7.

	Coverage penalty					**Coverage spread penalty**					**Overtime hours spread penalty**				
		SA		**VNS**			**SA**		**VNS**			**SA**		**VNS**	
Instance	**G**	**(1)**	**(2)**	**(1)**	**(2)**	**G**	**(1)**	**(2)**	**(1)**	**(2)**	**G**	**(1)**	**(2)**	**(1)**	**(2)**
O4	17,700	*	*	*	*	43,500	*	*	*	*	2,203.79	*	2,205.22	*	2,210.12
O5	22,800	*	*	*	*	72,800	*	*	*	*	1,488.96	*	1,491.65	*	1,498.03
O6	20,300	*	*	*	*	58,800	*	*	*	*	2,074.40	*	2,077.04	*	2,078.43
T4	3,600	*	*	*	*	0	*	*	*	*	1,205.07	*	*	*	*
T5	5,400	*	*	*	*	0	*	*	*	*	319.02	*	*	*	*
T6	8,000	*	*	*	*	600	*	*	*	*	1,185.63	*	*	*	*
V4	6,500	*	*	*	*	0	*	*	*	*	62.94	*	*	*	62.98
V5	4,200	*	*	*	*	0	*	*	*	*	105.31	*	106.61	*	106.56
V6	11,200	*	*	*	*	1,300	*	*	*	*	103.48	319.67	323.06	334.22	354.65

Table 7.9: Results obtained by Gurobi (G), compared with results of Simulated Annealing (SA) and Variable Neighborhood Search (VNS), applied both on the partial objective (1) and the complete objective (2), averaged over 10 runs per instance. The partial objective contains the coverage, coverage spread, overtime hours spread and requests penalties. The complete objective contains the partial objective and the additional objective, which consists of the preferred shift sequence length and preferred day-off sequence length penalties. For Gurobi, SA (1) and VNS (1), the additional objective and the total penalty were calculated afterwards for the resulting rosters. Note that the results from Gurobi form a lower bound for the partial objective. Asterisks (*) indicate that the SA or VNS result was equal to the value found by Gurobi for each run.

	Requests penalty					**Preferred shift sequence length penalty**					**Preferred day-off sequence length penalty**				
		SA		**VNS**			**SA**		**VNS**			**SA**		**VNS**	
Instance	**G**	**(1)**	**(2)**	**(1)**	**(2)**	**G**	**(1)**	**(2)**	**(1)**	**(2)**	**G**	**(1)**	**(2)**	**(1)**	**(2)**
O4	171	*	*	*	*	1,332.98	1,317.08	747.08	1,324.91	737.02	1,041.73	1,048.79	531.06	1,050.09	541.06
O5	220	*	*	*	*	1,563.87	1,573.63	786.99	1,578.74	791.45	1,214.46	1,196.47	525.44	1,207.71	536.56
O6	168	178	178	178	178	1,163.60	1,148.27	588.81	1,153.06	585.30	858.75	836.49	365.20	833.17	365.55
T4	0	*	6.6	*	5.5	1,005.12	981.33	561.98	977.44	576.01	738.94	712.05	488.49	705.03	484.92
T5	0	*	*	*	*	996.26	985.90	532.17	995.60	544.63	609.74	615.79	395.62	639.40	394.60
T6	0	*	*	*	*	994.81	989.24	519.73	980.99	531.47	586.42	590.17	351.10	580.73	353.30
V4	39	*	*	*	*	1,024.83	1,007.42	454.32	1,013.41	467.39	785.52	691.38	444.74	785.48	438.78
V5	77	*	*	*	*	1,084.96	1,103.56	484.05	1,113.89	492.20	842.64	711.63	436.24	849.24	438.18
V6	77	117	135.8	130.6	146.1	1,186.66	1,187.48	606.64	1,218.38	614.41	947.13	743.69	421.20	797.10	416.80

7.4. Comparison with ORTEC's algorithms

AutoRoster Figures 7.8 and D.10 and Tables D.15, D.16 and D.17 show the results of Simulated Annealing and Variable Neighborhood Search, compared to the AutoRoster local search algorithm, when applied to the problem described in Section 6.5. It can be seen that Simulated Annealing and Variable Neighborhood Search outperform AutoRoster on each instance. Note that the AutoRoster local search algorithm is mostly deterministic, and thus, no deviation can be observed among the results of different runs. In terms of penalty categories, AutoRoster mainly scores worse in the requests and preferred shift and day-off sequence length penalties.

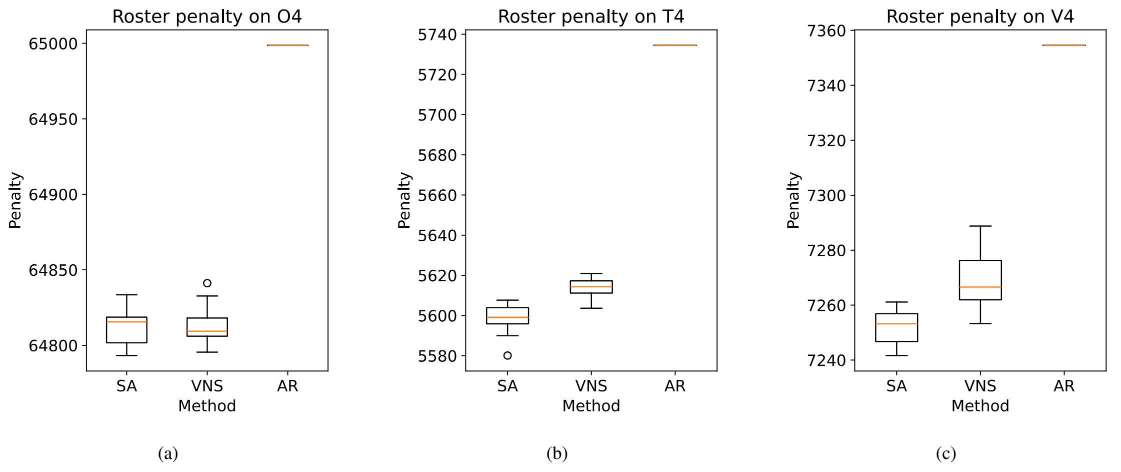


Figure 7.8: Box plots of the results of Simulated Annealing (SA) and Variable Neighborhood Search (VNS) compared to the AutoRoster (AR) local search algorithm on the modified model, shown in the AutoRoster column of Table 6.4, on the April instances over 10 runs per instance.

OWS solver Results of Gurobi, Simulated Annealing, AutoRoster and the OWS solver on the simplified model, as described in Section 6.5, on the trauma instances are shown in Tables 7.10 and 7.11. Because of the small differences in performance between Simulated Annealing and Variable Neighborhood Search, compared to their differences compared to AutoRoster, Variable Neighborhood Search was not included in this comparison. Simulated Annealing found the optimal solution on all three instances, AutoRoster on two out of three instances, and OWS on none of the instances. Specifically, all methods found the optimal values for the coverage penalty, AutoRoster had a higher overtime hours spread penalty once, and OWS had a higher overtime hours spread penalty twice, and a higher requests penalty once. Note that the differences are small. However, since the model was highly simplified, the qualitative difference in whether the methods were able to find the optimal solution gives some indication that Simulated Annealing could outperform OWS.

Table 7.10: Comparison of the results of Gurobi (G), Simulated Annealing (SA), the AutoRoster local search algorithm (AR) and the OWS solver on a simplified model for the trauma instances. Note that the results from Gurobi form a lower bound for the total penalty. Asterisks (*) indicate that the SA, AR or OWS result was equal to the value found by Gurobi.

	Total penalty				**Coverage penalty**				**Overtime hours spread penalty**			
Instance	**G**	**SA**	**AR**	**OWS**	**G**	**SA**	**AR**	**OWS**	**G**	**SA**	**AR**	**OWS**
T4	240,649	*	240,662	240,662	240,000	*	*	*	649	*	662	662
T5	320,820	*	*	320,891	320,000	*	*	*	720	*	*	791
T6	441,385	*	*	441,685	440,000	*	*	*	1,285	*	*	*

Table 7.11: Comparison of the results of Gurobi (G), Simulated Annealing (SA), the AutoRoster local search algorithm (AR) and the OWS solver on a simplified model for the trauma instances. Note that the results from Gurobi form a lower bound for the total penalty. Asterisks (*) indicate that the SA, AR or OWS result was equal to the value found by Gurobi.

	Requests penalty				**Unplanned shifts**				**Overtime hours per employee**				**Unsatisfied requests**			
Instance	**G**	**SA**	**AR**	**OWS**	**G**	**SA**	**AR**	**OWS**	**G**	**SA**	**AR**	**OWS**	**G**	**SA**	**AR**	**OWS**
T4	0	*	*	*	24	*	*	*	2.55	*	2.58	2.58	0	*	*	*
T5	100	*	*	*	32	*	*	*	3.30	*	*	3.43	1	*	*	*
T6	100	*	*	400	44	*	*	*	4.86	*	*	*	1	*	*	4

8 Discussion

In this chapter, we discuss some possible explanations of the obtained results, as well as the most important research limitations.

8.1. Differences between construction methods

For most of the instances, the difference in the obtained total penalty between Construction-per-shift and Construction-per-employee is explained for the largest part by the difference in the coverage spread penalty. We observed that in rosters generated by Construction-per-employee, the unplanned shifts are largely concentrated at the end of the scheduling period, which can be ascribed to the chronological order in which the employee rosters are constructed: once the final few days of the scheduling period are encountered, most employees will have already reached their maximum workload limit, due to their shift assignments on previous days, and cannot be assigned to any shift in those last few days. Consequently, the large number of unplanned shifts on only a small number of days results in a large coverage spread penalty. A similar observation was made during the process of determining the sorting criteria for Construction-per-shift, where applying the date criterion for sorting shifts resulted in worse performance than random sorting. Note that for both observations, the performance with assigning shifts in chronological order did not only lead to increased coverage spread penalty, but also for example more unplanned shifts and unsatisfied requests. From this, we conclude that constructing a roster in chronological order generally seems to have a negative effect on the quality of the solution.

In light of this conclusion, Construction-per-employee might perform better if it were adjusted, such that it goes through the days in a non-chronological manner for each employee. However, it was noted that Construction-per-shift, in the case where the shifts were only sorted by date and the employees were sorted completely randomly, still outperformed the best found variant of Construction-per-employee on 8 out of 9 instances. This is an indication that constructing a roster per employee generally leads to worse solutions than constructing it per shift. Therefore, no additional tests were performed with a non-chronological version of Construction-per-employee.

8.2. Effect of different neighborhood structures

In the iterations of Algorithm 8, most clearly in the first iteration, it was noted that the different neighborhood structures contributed to improvements in different aspects of the objective function. Firstly, the coverage spread penalty was improved the most by UC2 and H1. This can be explained by their ability to alter the distribution of the existing shifts over the different days, without adding new shifts, and such that the employee workloads remain the same. Secondly, the overtime hours spread penalty was improved the most by V1. This can be explained by its ability to move shifts between employees, such that the worked overtime is distributed more evenly among them, without altering shift coverage. Finally, since UC1 and C1 are the only neighborhood structures that are able to increase the total number of shifts in the roster, the lowest found values for the coverage penalty were reached only after adding either one of them. By design of the construction methods and UC1, UC1 on itself cannot improve the coverage penalty of the initial solution, as all unplanned shifts in the initial solution cannot be directly assigned to any employee without causing a hard constraint violation. Therefore, UC1 can only improve the roster penalty if combined with other neighborhood structures, whereas C1 could find the lowest found coverage penalty values on its own both for Variable Neighborhood Search and Simulated Annealing.

The observations described above explain the results shown in Table 7.3, namely that horizontal swapping, vertical swapping and shift assignment changing operations are all necessary to reach the best overall performance.

8.3. Overplanned shifts

During the execution of Algorithm 8, it was observed that the addition of C1 caused a great increase in the number of overplanned shifts in the final roster for the instances of the obstetrics and vascular surgery departments. In these instances, employees can be identified who only have the required skills for shifts for which the coverage requirements are relatively low. When those coverage requirements are met, such employees have sufficient time left within their maximum workload limit to be assigned to additional shifts. Although these additional shifts are redundant with respect to the coverage requirements, they may improve the solution with respect to employee preference constraints. Note that due to the overtime hours penalty, it is not beneficial to plan much overtime using redundant shifts. In the instances of the trauma department, there are no employees with time left for additional redundant shifts, and thus, no overplanned shifts are assigned. Beforehand, it was expected that the problem instances would not allow for the assignment of many non-required shifts, as understaffing is generally more common in healthcare than overstaffing. Therefore, no maximum shift coverage constraints were added to the problem. However, it can be viewed as undesirable if many redundant shifts are planned. To account for this, neighborhood restrictions were added that prevent assigning shifts of which the coverage requirement is zero on the day in question, as described in Section 5.2. Resultingly, overcoverage could only occur for shifts for which the coverage requirement is at least one. We would advise to let maximum shift coverage constraints be optional in practice, such that it can be used or not, depending on whether overcoverage is likely and/or desirable to happen.

8.4. Effect of maximum block length

The difference between Simulated Annealing and Variable Neighborhood Search in their results with variable maximum block lengths, shown in Section 7.2.1, is noteworthy. For both frameworks, increasing the maximum block lengths initially improved performance. However, this trend reversed for Simulated Annealing around a maximum block length of 4, whereas it continued for Variable Neighborhood Search for greater maximum block lengths, until it eventually stagnated. For Simulated Annealing, it was observed that operations sampled from neighborhood structures with greater block lengths typically have a lower acceptance rate than those sampled from neighborhood structures with smaller block lengths, as is expressed in the probabilities shown in Table 7.5. For Variable Neighborhood Search, similar observations were made. Thus, operations with greater block lengths are more likely to be ineffective, i.e. cause infeasibility and/or increase the roster penalty.

Recall that for Simulated Annealing, all neighborhood structures had equal sampling probabilities in the experiments with different maximum block lengths. For Simulated Annealing, this likely means that, for increasing maximum block lengths, the disadvantage of increasing the number of ineffective operations eventually outweighs the benefit of expanding the set of possible operations that can be sampled. The results with different neighborhood probability settings, as described in Section 7.2.3, show that assigning smaller probabilities to neighborhood structures with greater block lengths results in increased performance. This suggests that the addition of neighborhood structures with block lengths greater than 4 could still improve performance, if the sampling probabilities are adjusted accordingly.

For Variable Neighborhood Search, the addition of neighborhood structures with greater block lengths results in a longer search for a local minimum in the descent phase. Although a neighborhood with a very large block length might contain only very few improving operations, every time that such a neighborhood contains at least one improving operation triggers a restart of the descent phase as a whole, potentially resulting in a better local minimum. A consequence of a longer descent phase is that fewer perturbations to escape local minima can be made within the same time limit. The results described in Section 7.2.2 showed that, generally, a larger maximum block length leads to improved performance for Variable Neighborhood Search. Thus, this likely means that the benefit in terms of exploitation, through a longer descent phase with relatively more ineffective operations, outweighs the benefit in terms of diversification, through more perturbations to escape local minima. Note however, that the addition of neighborhood structures with greater block lengths also leads to improved diversification within the descent phase, as more different solutions can be reached by expanding the set of considered operations. In conclusion, these results suggest that the perturbation is not very effective, potentially because there are not many different local minima or the existing local minima have very similar objective function values, which would reduce the relevance of diversification, or the used perturbation method does not manage to jump to significantly different locations in the solution space.

8.5. Effect of overall framework

The results of the comparison of Simulated Annealing and Variable Neighborhood Search, shown in Section 7.2.5, show that Simulated Annealing generally outperforms Variable Neighborhood Search. However, the differences are relatively small. Thus, additional adjustments to and/or parameter tuning for the algorithms could still change this relation to some extent. Also, when we compare the results on the different penalty categories, the differences remain small, and no tradeoff can be observed, where Simulated Annealing scores better on some categories, and Variable Neighborhood Search on others. Note that the algorithms are quite different in how they use their sets of neighborhood structures, as Simulated Annealing is mostly probabilistic, and Variable Neighborhood Search is much more structured. However, the sets of used neighborhood structures of both methods largely overlap. Thus, these results suggest that the used types of roster operations have a much greater influence on the overall performance than the framework in which they are applied.

8.6. Lower bounds

The results shown in Section 7.3 show that Simulated Annealing and Variable Neighborhood Search could attain the lower bounds on the partial objective obtained by Gurobi in each run for 7 out of 9 instances, even though Gurobi used a model where the hard constraints on weekly rest, rest after consecutive night shifts and consecutive shifts with night shifts were relaxed. This shows that the addition of these hard constraints that were relaxed for Gurobi did not increase the optimal partial objective values of the problem instances. Thus, although these hard constraints are necessary to ensure that solutions satisfy the Dutch Working Hours Act, in most cases they do not reduce the extent to which the soft constraints from the partial objective can be satisfied.

When we compare the standard deviation of the results of Simulated Annealing and Variable Neighborhood Search, shown in Table D.11, with the differences between the lower bounds on the partial objective and the results of Simulated Annealing and Variable Neighborhood Search on the partial objective, as shown in Table 7.7 (columns G, SA (2) and VNS (2) for the partial objective), we find that for most instances, the standard deviation is greater than this difference in partial objective. Additionally, we observe that for both Simulated Annealing and Variable Neighborhood Search, the standard deviations in the preferred shift and day-off sequence length penalties were significantly higher than the standard deviations in the coverage, coverage spread, overtime hours spread and requests penalties for most instances. This suggests that the greatest part of the standard deviation in the results of Simulated Annealing and Variable Neighborhood Search can be ascribed to the preferred shift and day-off sequence length penalties. This could mean that the differences in quality of different local minima originate mostly from these soft constraints. We can at least partly explain this by the fact that there are often multiple ways to improve these soft constraint penalties. For example, both penalties can be decreased by either changing at least one shift/day-off sequence into the preferred length, or by merging two shift/day-off sequences that are not of the preferred length into one, as illustrated in Figure 8.1. Note that the latter option can give counterintuitive results, as the creation of sequences, that deviate even more from the preferred length than the current sequences, can improve the penalty. As illustrated in Figure 8.1, changing one sequence into the preferred length improves the penalty more than merging two sequences into one. However, our methods do not employ a steepest-descent principal, and thus, it is possible to improve the penalty by repeatedly merging sequences, resulting in rosters with very long sequences. This phenomenon could cause the existence of many different local minima, with different combinations of very long sequences and sequences of the preferred length, which could at least partly explain the relatively large deviation in the preferred shift and day-off sequence length penalties.

8.7. Comparison with ORTEC's algorithms

The results of the comparison with the AutoRoster local search algorithm and OWS suggest that Simulated Annealing and Variable Neighborhood Search outperform both AutoRoster and OWS. However, the comparison with OWS was limited to a simplified model on only three instances. Therefore, more extensive comparisons on the complete model are required for more strongly founded conclusions.

8.8. Research limitations

In this study, different variations of several components of the used methods were tested, such as the shift sorting criteria for the construction methods, or the maximum block lengths of the neighborhoods for the overall frameworks. Because the total number of combinations of all variations of these components was too large to test them all, the components were varied one at a time, and the best variation of one component was fixed before testing variations of the next component. However, the different algorithm components are likely interdependent

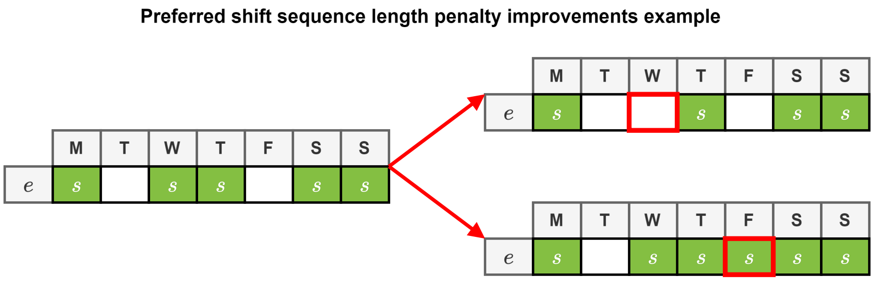

Preferred shift sequence length penalty improvements example
M
T
W
T
F
S
S
e
s

Figure 8.1: Example of possible ways to improve the preferred shift sequence length penalty. Suppose employee e has some assignments of shift type s, and prefers shift sequences of length 1. Then, in the starting roster on the left, there is one shift sequence of the preferred length, and two not of the preferred length. Thus, the ratio of shift sequences that are *not* of the preferred length is $\frac{2}{3}$. On the right, two possible ways to improve the corresponding penalty are shown. In the top case, one shift is removed, which results in one additional shift sequence to attain the preferred shift sequence length, thereby decreasing the ratio of sequences not of the preferred length to $\frac{1}{3}$. In the bottom case, one shift is added, which results in the reduction of the number of shift sequences by one, while keeping the number of shift sequences of the preferred length the same. Thereby, the ratio of sequences not of the preferred length is decreased to $\frac{1}{2}$.

to some extent, such that variations in one component can affect the influence of variations in another component. Therefore, there might be combinations of component variations that would result in better performance than the combinations that were the result of this study. An example could be the relation between the maximum block length and the neighborhood probability distribution for Simulated Annealing, as described in Section 8.4.

Furthermore, we aimed for methods that are general, such that they can perform well under various problem setting and data instances. However, only one problem setting and data instances from one hospital were used. Although these were chosen to be representative, still, the generality of the methods would have to be validated by testing them in different contexts.

Additionally, the conclusions on some experiments had to be made based on very small differences among the results of different algorithm component variations. Altogether, to draw more strongly founded conclusions on which specific algorithm configurations perform best, tests should be done with more variations in problem descriptions and data instances, with more runs per instance, and with more finely tuned components, based on the general trends that our results exhibit.

9 Conclusion

This study was aimed at finding an algorithm that is suitable for implementation in a commercial automatic shift scheduler. Firstly, the state-of-the-art in algorithms to solve the nurse rostering problem was identified: literature on the nurse rostering problem was reviewed, algorithms presented in literature were identified, and their results on different benchmarks were compared. In general, single-solution based meta-heuristics were deemed the most suitable for the target cases, based on their performance in terms of solution quality and reliability in terms of finding feasible solutions. A categorization was made of the components that such algorithms typically consist of, such as a construction methods, neighborhood structures, overall frameworks and perturbation methods.

Based on the conclusions from the literature review, we chose to implement two construction methods, namely Construction-per-shift and Construction-per-employee, and two overall frameworks, namely Simulated Annealing and Variable Neighborhood Search. Variations of several components of these methods were tested: different shift and employee sorting criteria for the construction methods, different types of neighborhood structures for the overall frameworks, different maximum block lengths of the used neighborhood structures, different neighborhood probabilities for Simulated Annealing and different neighborhood orderings for Variable Neighborhood Search.

Experiments were performed under a time limit of 10 minutes using nine data instances from a Dutch hospital, originating from three departments over three months. The considered hard and soft constraints were drawn up to reflect real-world target cases.

The best performing construction method was Construction-per-shift, where shifts were sorted based on the shift type priority and coverage requirements, and employees were sorted based on day and shift requests, and the amount of available working time. Construction-per-shift greatly outperformed the best performing variant of Construction-per-employee.

The best performing overall framework was Simulated Annealing, using horizontal swapping, vertical swapping and shift assignment changing neighborhood structures with block lengths 1 through 4. Generally, a higher neighborhood probability was given to vertical swapping neighborhoods than to horizontal swapping and shift assignment changing neighborhoods, and a lower probability to neighborhoods with a larger block length. In terms of performance, Simulated Annealing had a small advantage compared to Variable Neighborhood Search, for which the best performing variant used two undercoverage focused neighborhood structures, and horizontal swapping, vertical swapping, and shift assignment changing neighborhood structures of block lengths 1 through 10, 1 through 19 and 1 through 7, respectively.

Based on the obtained results, we evaluate the suitability of the implemented methods against the required characteristics:

- **stability:** the combined methods of Construction-per-shift and Simulated Annealing, and Construction-per-shift and Variable Neighborhood Search were able to produce stable results over multiple runs, with standard deviations of only 0.077% and 0.085% of the mean objective function value, respectively, averaged over the nine instances.

- **generality:** Construction-per-shift uses problem information such as the shift type priorities, coverage requirements and employee requests. Therefore, its performance will likely be affected by differences in the problem formulation. In contrast, Simulated Annealing, of which the performance was shown to be largely independent of the construction method, only employs basic roster operations of swapping and changing shift assignments, that do not depend on the specific problem information. For Variable Neighborhood

Search, only the undercoverage focused neighborhood structures use problem information. Therefore, we expect these methods to perform well for a wide variety of hard and soft constraints within the nurse rostering problem, although tuning of, for example, temperature parameters and neighborhood probabilities might still be required in different contexts.

- **high quality rosters:** with respect to the combined shift coverage, shift coverage spread, overtime hours spread and requests penalties, Simulated Annealing and Variable Neighborhood Search obtained near-optimal results. For the preferred shift and day-off sequence length penalties, no lower bound was known, but significant improvements were made compared to the constructed initial roster. These results indicate that Simulated Annealing and Variable Neighborhood Search can produce rosters of high quality, in terms of both coverage requirements and employee preferences.
- **short computation times:** the results were obtained using a time limit of 10 minutes, which can already be considered as reasonably short. Further improvements in the implementation efficiency can ensure that the same solution quality can be obtained in even less time.

We conclude that both Simulated Annealing and Variable Neighborhood Search are suitable methods for implementation in a general automatic shift scheduler, where Simulated Annealing holds a small advantage in terms of performance.

10 Future research

In this chapter, we discuss several research directions that could be addressed in future studies.

10.1. Algorithm variations

In this study, different variations of several components of the used methods were tested, such as the shift sorting criteria for the construction methods, or the probability distributions of the neighborhoods for Simulated Annealing. However, many more algorithm adaptations could be experimented with. We list several suggestions for possible improvements of the algorithms:

- **Non-hierarchichal sorting of shifts and employees:** in the construction methods, shift and employee sorting criteria were used, to try to schedule the most difficult shifts and employees first, and take some soft constraints into account. We applied these sorting criteria hierarchichally. Thus, if one shift or employee was ranked as more difficult than another with respect to the first criterion, it was always placed earlier in the list, independent of how they scored on the other criteria. Only if two shifts or employees were equal with respect to the first criterion, they were sorted by the second criterion, etc. However, suppose shift s_1 is ranked as more difficult than shift s_2 on all criteria, except for the first. Then, shift s_2 would be tried to assign first based on the first criterion, even though the combination of the other criteria might make shift s_1 more difficult to assign than shift s_2. In order to sort the shift and employees based on all criteria non-hierarchichally, a difficulty score could be used, equal to a weighted sum of the scores on the different criteria. The weights would then signify how much the different criteria contribute to the difficulty of assigning a shift or employee.

- **Less randomized perturbation:** in Variable Neighborhood Search, the descent and perturbation phases are alternated repeatedly, where the perturbation phase is started once the descent phase has yielded a local minimum. In our study, the perturbation phase consisted of applying randomly sampled vertical swapping, horizontal swapping and shift assignment changing operations, until the roster penalty was changed by at least 5%. However, this might not be the most effective way to perturb the solution. Firstly, the random perturbation moves could be replaced by, or combined with a ruin-and-recreate method, where the rosters on one or more days or employees could be (partially) removed and then reconstructed. The reconstruction could, for example, be done similarly to the construction methods used in this study. Secondly, the perturbation method could be applied in a targeted fashion, such that, for example, the days or employees with the highest penalties are perturbed more frequently.

- **Adaptive neighborhood probabilities:** in Simulated Annealing, each neighborhood structure requires a specified probability for it to be sampled in each iteration. We tested several different probability distributions for the set of neighborhood structures used in Simulated Annealing, such that the neighborhood probabilities were the same for every iteration, and we obtained a distribution that resulted in better performance than the default of equal probabilities for all neighborhoods. However, it might be the case that some neighborhood structures are very effective in the early stages of solution improvement, but less effective in the later stages, and vice versa for other neighborhood structures. Therefore, it could be beneficial to adjust the neighborhood probabilities adaptively over time. For example, the probabilities could be determined based on similar statistics as described in Section 6.3.3, but then adjusted after each iteration. Note that such adaptive neighborhood probabilities show similarities with the hidden Markov model-based hyper-heuristic by Kheiri et al. (2021).

- **Parallel runs:** the overall frameworks of Simulated Annealing and Variable Neighborhood Search were only tested with a time limit of 10 minutes. It would be insightful to test them also with different time limits, to find the relation between their performance and the used time limit. For example, suppose that Simulated Annealing exhibits the potential to reach the same solution quality in half the time, but only reaches it less frequently within that shorter time limit. Then, it might be beneficial to execute two runs in parallel with half the available computation time each, and take the best result of the two, than to execute one run with all available computation time.

10.2. Hard constraint relaxations

The hard constraints of the problem reflect the criteria that a roster must satisfy for it to be acceptable, independent of its quality in terms of the soft constraints. Therefore, the roster that an algorithm returns as output must satisfy all hard constraints. However, allowing intermediate solutions to be infeasible may enable an algorithm to access feasible regions that would otherwise be hard to reach, potentially leading to better final solutions. Such relaxations can be implemented by changing the hard constraints into soft constraints with specified weights. The weights should then be sufficiently high, compared to those of the soft constraints, in order for algorithms to steer towards feasible solutions. This is because too low weights may cause an algorithm to favor satisfying a few soft constraints over satisfying a hard constraint. On the other hand, if the relaxed hard constraints have weights that are very high, they can remain unrelaxed in practice, depending on the overall framework. In the Simulated Annealing framework, weights that are too high will result in negligible acceptance probabilities for moves towards rosters that violate any of the relaxed hard constraints. In the Variable Neighborhood Search framework, improving moves that violate any relaxed hard constraints are then unlikely to be found within the descent phase. However, in the perturbation phase, moves are accepted independent of their effect on the roster penalty, as long as they are feasible, so in the perturbation phase, the relaxations will be effective independent of the applied weights.

Hard constraint violations differ in how difficult they are to resolve. For example, if a shift is assigned to an employee who does not have the required skills, that same shift has to be unassigned again later to resolve the violation. In contrast, a maximum workload violation can be (partly or completely) resolved by unassigning any of the currently assigned shifts for the employee in question. Thus, an interesting direction for future research would be to study if improvements in algorithm performance can be obtained by hard constraint relaxations, where the weights reflect the difficulty of resolving the corresponding violations.

10.3. Exact methods

In this study, a simplified model was solved using an exact method, in order to obtain lower bounds to a partial objective, consisting of 4 out of 6 soft constraint penalties. However, solving the complete model using an exact method was outside of the scope of this project, because of the expected complexity of the problem, both in terms of the required amount of time to formulate the mixed-integer program, and in terms of the computational time and memory required to solve it. However, being able to find the exact solutions to problem instances with respect to the total roster penalty could give additional insight in the optimality gaps that the algorithms can reach. Note that for the partial objective, omitting the hard constraints on weekly rest, rest after consecutive night shifts, and consecutive shifts with night shifts hard constraints did not prevent the lower bound being reached for most problem instances, as described in Section 7.3. Thus, only the addition of the preferred shift and day-off sequence length penalties to the simplified model could already be enough to obtain strong lower bounds to the total roster penalty.

10.4. Implementation efficiency

For optimization methods in general, good performance requires not only an effective algorithm, but also an efficient implementation. In the considered problem, calculations of the hard constraint violations and soft constraint penalties take up most of the computation time. Therefore, reducing the amount of redundant penalty calculations can significantly reduce the computation time per iteration, thereby improving the solution quality that can be reached within the same amount of time. Specifically, for each type of roster operation, it should be determined which constraints have to be recalculated, and which constraints are not affected, after applying it. For example, if a horizontal swap is applied within the roster of an employee, the total amount of working time for that employee cannot have changed, and thus the maximum workload constraint does not have to be recalculated. Similarly, if a vertical swap is applied on a day in the roster, the total shift coverage on that day cannot have changed, and thus the coverage and coverage spread penalties do not have to be recalculated.

Thus, although we already obtained good results and drew several conclusions on the performance of different algorithm components, many more promising developments can be expected for future solutions to the nurse rostering problem.

Bibliography

Abdelghany, M., Eltawil, A. B., Yahia, Z., & Nakata, K. (2021a). A hybrid variable neighbourhood search and dynamic programming approach for the nurse rostering problem. *Journal of Industrial and Management Optimization*, *17*(4), 2051–2072. https://doi.org/10.3934/jimo.2020058

Abdelghany, M., Yahia, Z., & Eltawil, A. B. (2021b). A new two-stage variable neighborhood search algorithm for the nurse rostering problem. *RAIRO-Operations Reasearch*, *55*(2), 673–687. https://doi.org/10.1051/ro/2021027

Abuhamdah, A., Boulila, W., Jaradat, G. M., Quteishat, A. M., Alsmadi, M. K., & Almarashdeh, I. A. (2021). A novel population-based local search for nurse rostering problem. *International Journal of Electrical and Computer Engineering*, *11*(1), 2051–2072. https://doi.org/10.11591/ijece.v11i1.pp471-480

Arbeidstijdenwet. (2022). §5.2 [Accessed on June 3 2024]. https://wetten.overheid.nl/BWBR0007671/2022-08-02/0#Hoofdstuk5

Awadallah, M. A., Al-Betar, M. A., Khader, A. T., Bolaji, A. L., & Alkoffash, M. (2017). Hybridization of harmony search with hill climbing for highly constrained nurse rostering problem. *Neural Computing and Applications*, *28*, 463–482. https://doi.org/10.1007/s00521-015-2076-8

Awadallah, M. A., Bolaji, A. L., & Al-Betar, M. A. (2015). A hybrid artificial bee colony for a nurse rostering problem. *Applied Soft Computing*, *35*, 726–739. https://doi.org/10.1016/j.asoc.2015.07.004

Awadallah, M. A., Khader, A. T., Al-Betar, M. A., & Bolaji, A. L. (2014). Harmony search with novel selection methods in memory consideration for nurse rostering problem. *Asia-Pacific Journal of Operational Research*, *31*(3), 1450014. https://doi.org/10.1142/S0217595914500146

Burke, E. K., & Curtois, T. (2014). New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, *237*(1), 71–81. https://doi.org/10.1016/j.ejor.2014.01.039

Burke, E. K., Curtois, T., Post, G., Qu, R., & Veltman, B. (2008). A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, *188*(2), 330–341. https://doi.org/10.1016/j.ejor.2007.04.030

Burke, E. K., Curtois, T., Qu, R., & Vanden Berghe, G. (2013). A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing*, *25*(3), 411–419. https://doi.org/10.1287/ijoc.1120.0510

Burke, E. K., De Causmaecker, P., Vanden Berghe, G., & Van Landeghem, H. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, *7*, 441–499. https://doi.org/10.1023/B:JOSH.0000046076.75950.0b

Burke, E. K., Li, J., & Qu, R. (2010). A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, *203*(2), 484–493. https://doi.org/10.1016/j.ejor.2009.07.036

CBS. (2022). Werkdruk en arbeidstevredenheid in de zorg [Accessed on June 3 2024]. https://www.cbs.nl/nl-nl/longread/statistische-trends/2022/werkdruk-en-arbeidstevredenheid-in-de-zorg

CBS. (2024). Ziekteverzuimpercentage; arbeidsmarkt zorg en welzijn [Accessed on June 3 2024]. https://azwstatline.cbs.nl/#/AZW/nl/dataset/24015NED/table?dl=57BDF

Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S., & Schaerf, A. (2019). The second international nurse rostering competition. *Annals of Operations Research*, *274*(1), 171–186. https://doi.org/10.1007/s10479-018-2816-0

Ceschia, S., Dang, N. T. T., De Causmaecker, P., Haspeslagh, S., & Schaerf, A. (2015). Second international nurse rostering competition (inrc-ii), problem description and rules. https://arxiv.org/abs/1501.04177

Ceschia, S., Di Gaspero, L., Mazzaracchio, V., Policante, G., & Schaerf, A. (2023). Solving a real-world nurse rostering problem by simulated annealing. *Operations Research for Health Care*, *36*, 100379. https://doi.org/10.1016/j.orhc.2023.100379

Ceschia, S., Guido, R., & Schaerf, A. (2020). Solving the static inrc-ii nurse rostering problem by simulated annealing based on large neighborhoods. *Annals of Operations Research*, *288*(1), 95–113. https://doi.org/10.1007/s10479-020-03527-6

Chen, Z., De Causmaecker, P., & Dou, Y. (2023). A combined mixed integer programming and deep neural network-assisted heuristics algorithm for the nurse rostering problem. *Applied Soft Computing*, *136*, 109919. https://doi.org/10.1016/j.asoc.2022.109919

Curtois, T., & Qu, R. (2014). Computational results on new staff scheduling benchmark instances. *ASAP Res. Group, School Comput. Sci., Univ. Nottingham, Nottingham, UK, Tech. Rep.* http://www.schedulingbenchmarks.org/papers/computational_results_on_new_staff_scheduling_benchmark_instances.pdf

Den Hartog, S. J., Hoogeveen, H., & Van der Zanden, T. C. (2023). On the complexity of nurse rostering problems. *Operations Research Letters*, *51*(5), 483–487. https://doi.org/10.1016/j.orl.2023.07.004

Goh, S. L., Sze, S. N., Sabar, N. R., Abdullah, S., & Kendall, G. (2022). A 2-stage approach for the nurse rostering problem. *IEEE Access*, *10*, 69591–69604. https://doi.org/10.1109/ACCESS.2022.3186097

Hadwan, M., Ayob, M., Sabar, N. R., & Qu, R. (2013). A harmony search algorithm for nurse rostering problems. *Information Sciences*, *233*, 126–140. https://doi.org/10.1016/j.ins.2012.12.025

Haspeslagh, S., De Causmaecker, P., Schaerf, A., & Stølevik, M. (2014). The first international nurse rostering competition 2010. *Annals of Operations Research*, *218*, 221–236. https://doi.org/10.1007/s10479-012-1062-0

Haspeslagh, S., De Causmaecker, P., Stølevik, M., & Schaerf, A. (2010). First international nurse rostering competition 2010. https://nrpcompetition.kuleuven-kulak.be/wp-content/uploads/2020/06/nrpcompetition_description.pdf

Helder, C. (2023). Arbeidsmarktprognose zorg en welzijn 2023 [Letter from Minister of Health, Welfare and Sport to Dutch parliament]. https://www.rijksoverheid.nl/documenten/kamerstukken/2023/12/21/kamerbrief-over-arbeidsmarktprognose-zorg-en-welzijn-2023

Jaradat, G. M., Al-Badareen, A., Ayob, M., Al-Smadi, M., Al-Marashdeh, I., Ash-Shuqran, M., & Al-Odat, E. (2019). Hybrid elitist-ant system for nurse-rostering problem. *Journal of King Saud University - Computer and Information Sciences*, *31*(3), 378–384. https://doi.org/10.1016/j.jksuci.2018.02.009

Kheiri, A., Gretsista, A., Keedwell, E., Lulli, G., Epitropakis, M. G., & Burke, E. K. (2021). A hyper-heuristic approach based upon a hidden markov model for the multi-stage nurse rostering problem. *Computers and operations research*, *130*, 105221. https://doi.org/10.1016/j.cor.2021.105221

Lan, S., Fan, W., Yang, S., Pardalos, P. M., & Mladenovic, N. (2021). A survey on the applications of variable neighborhood search algorithm in healthcare management. *Annals of Mathematics and Artificial Intelligence*, *89*, 741–775. https://doi.org/10.1007/s10472-021-09727-5

Legrain, A., Omer, J., & Rosat, S. (2020a). A rotation-based branch-and-price approach for the nurse scheduling problem. *Mathematical Programming Computation*, *12*, 417–450. https://doi.org/10.1007/s12532-019-00172-4

Legrain, A., Omer, J., & Rosat, S. (2020b). A rotation-based branch-and-price approach for the nurse scheduling problem. *Mathematical Programming Computation*, *285*, 417–450. https://doi.org/10.1016/j.ejor.2018.09.027

Lü, Z., & Hao, J.-K. (2012). Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, *218*(3), 865–876. https://doi.org/10.1016/j.ejor.2011.12.016

Meignan, D., & Knust, S. (2019). A neutrality-based iterated local search for shift scheduling optimization and interactive reoptimization. *European Journal of Operational Research*, *279*(2), 320–334. https://doi.org/10.1016/j.ejor.2019.06.005

Ngoo, C. M., Goh, S. L., Sabar, N. R., Abdullah, S., Kendall, G., et al. (2022). A survey of the nurse rostering solution methodologies: The state-of-the-art and emerging trends. *IEEE Access*, *10*, 56504–56524. https://doi.org/10.1109/ACCESS.2022.3177280

Özder, E. H., Özcan, E., & Eren, T. (2020). A systematic literature review for personnel scheduling problems. *International Journal of Information Technology & Decision Making*, *19*(06), 1695–1735. https://doi.org/10.1142/S0219622020300050

Rahimian, E., Akartunalı, K., & Levine, J. (2017). A hybrid integer programming and variable neighbourhood search algorithm to solve nurse rostering problems. *European Journal of Operational Research*, *258*(2), 411–423. https://doi.org/10.1016/j.ejor.2016.09.030

Römer, M., & Mellouli, T. (2016). A direct milp approach based on state-expanded network flows and anticipation for multi-stage nurse rostering under uncertainty. *Proceedings of the 11th international conference on the practice and theory of automated timetabling*, 549–551. https://www.patatconference.org/patat2016/files/proceedings/paper_56.pdf

Rosenström, T., Härmä, M., Kivimäki, M., Ervasti, J., Virtanen, M., Hakola, T., Koskinen, A., & Ropponen, A. (2021). Patterns of working hour characteristics and risk of sickness absence among shift-working hospital employees: A data-mining cohort study. *Scandinavian Journal of Work, Environment & Health*, *47*(5), 395–403. https://doi.org/10.5271/sjweh.3957

Strandmark, P., Qu, Y., & Curtois, T. (2020). First-order linear programming in a column generation-based heuristic approach to the nurse rostering problem. *Computers & Operations Research*, *120*, 104945. https://doi.org/10.1016/j.cor.2020.104945

Tassopoulos, I. X., Solos, I. P., & Beligiannis, G. N. (2015). A two-phase adaptive variable neighborhood approach for nurse rostering. *Computers & operations research*, *60*, 150–169. https://doi.org/10.1016/j.cor.2015.02.009

Turhan, A. M., & Bilgen, B. (2020). A hybrid fix-and-optimize and simulated annealing approaches for nurse rostering problem. *Computers & Industrial Engineering*, *145*, 106531. https://doi.org/10.1016/j.cie.2020.106531

Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., & Housos, E. (2012). A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, *219*(2), 425–433. https://doi.org/10.1016/j.ejor.2011.12.042

Van Rooijen, E. (2023). *Incorporating nurse preferences in the nurse scheduling problem* [Master's thesis, Erasmus University Rotterdam]. http://hdl.handle.net/2105/70265

Xu, S., & Hall, N. G. (2021). Fatigue, personnel scheduling and operations: Review and research opportunities. *European Journal of Operational Research*, *295*(3), 807–822. https://doi.org/10.1016/j.ejor.2021.03.036

Zheng, Z., Liu, X., & Gong, X. (2017). A simple randomized variable neighbourhood search for nurse rostering. *Computers & Industrial Engineering*, *110*, 165–174. https://doi.org/10.1016/j.cie.2017.05.027

A
Benchmark results

This appendix contains the results of algorithms found in literature, on scientific benchmarks, as described in Chapter 2.

Table A.1: Overview of algorithms and the abbreviations used in the tables with their experimental results

Authors	**Method**	**Abbreviation**
Abdelghany et al. (2021a)	Variable Neighborhood Search + Dynamic Programming	ABD21
Abdelghany et al. (2021b)	Two-stage Variable Neighborhood Search	ABD21b
Abuhamdah et al. (2021)	Population-Based Local Search	ABU21
Awadallah et al. (2015)	Bee Colony Optimization	AWA15
Awadallah et al. (2017)	Harmony Search	AWA17
Burke and Curtois (2014)	Variable Depth Search	BUR14
Chen et al. (2023)	Integer Programming + Neural Network assisted heuristic search	CHE23
Goh et al. (2022)	Monte Carlo Tree Search + Iterated Local Search	GOH22
Jaradat et al. (2019)	Elitist Ant Colony Optimization	JAR19
Lü and Hao (2012)	Adaptive Variable Neighborhood Search	LÜ12
Rahimian et al. (2017)	Integer Programming + Variable Neighborhood Search	RAH17
Tassopoulos et al. (2015)	Two-phase Adaptive Variable Neighborhood Search	TAS15
Turhan and Bilgen (2020)	Fix-and-Relax + Simulated Annealing + Fix-and-Optimize	TUR20
Valouxis et al. (2012)	Two-phase Integer Programming approach	VAL12
Zheng et al. (2017)	Randomized Variable Neighborhood Search	ZHE17

Table A.2: Best obtained results to the *sprint* track of the INRC-I benchmark. Best known solutions (BKS) are shown in bold if they are known to be optimal. Algorithm results are shown in bold if they equal the BKS. *AWA17 used an iteration number as stopping criterion and exceeded the allowed time limit in several *sprint* instances. **JAR19 used an iteration number as stopping criterion and did not report the used computation time. ***ABU21 reported using relaxed timeout conditions of 1000 seconds for the *sprint* track.

Instance	BKS	BUR14	VAL12	LÜ12	TAS15	AWA15	AWA17*	ZHE17	JAR19**	ABD21	ABU21***
sprint early 01	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	57	**56**	57
sprint early 02	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	59	**58**	**58**
sprint early 03	**51**	**51**	**51**	**51**	**51**	**51**	**51**	**51**	**51**	**51**	**51**
sprint early 04	**59**	**59**	**59**	**59**	**59**	**59**	**59**	**59**	**59**	**59**	**59**
sprint early 05	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**	**58**
sprint early 06	**54**	**54**	**54**	**54**	**54**	**54**	**54**	**54**	54	**54**	**54**
sprint early 07	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**
sprint early 08	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**	**56**
sprint early 09	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**	**55**
sprint early 10	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**	**52**
sprint late 01	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**	**37**
sprint late 02	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**
sprint late 03	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**	**48**
sprint late 04	**73**	75	76	**73**	**73**	**73**	**73**	**73**	**73**	**73**	**73**
sprint late 05	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	**44**	45	**44**
sprint late 06	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	**42**	43	**42**
sprint late 07	**42**	**42**	43	44	44	44	44	43	**42**	47	**42**
sprint late 08	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
sprint late 09	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**
sprint late 10	**43**	**43**	44	**43**	**43**	**43**	**43**	**43**	**43**	44	**43**
sprint hidden 01	**32**		33	**32**	**32**	**32**	**32**	**32**	**32**	34	**32**
sprint hidden 02	**32**		33	**32**	**32**	**32**	**32**	**32**	**32**	**32**	**32**
sprint hidden 03	**62**		**62**	**62**	**62**	**62**	**62**	**62**	**62**	**62**	**62**
sprint hidden 04	**66**		67	**66**	**66**	**66**	**66**	**66**	**66**	67	**66**
sprint hidden 05	**59**		60	**59**	**59**	**59**	**59**	**59**	**59**	**59**	**59**
sprint hidden 06	**130**		139	**130**	**130**	**130**	**130**		**130**	141	**130**
sprint hidden 07	**153**		**153**	**153**	**153**	**153**	**153**		**153**	**153**	**153**
sprint hidden 08	**204**		220	**204**	**204**	**204**	**204**		**204**	**204**	**204**
sprint hidden 09	**338**		**338**	**338**	**338**	**338**	**338**		**338**	**338**	**338**
sprint hidden 10	**306**		**306**	**306**	**306**	**306**	**306**		**306**	**306**	**306**

Table A.3: Best obtained results to the *medium* track of the INRC-I benchmark. Best known solutions (BKS) are shown in bold if they are known to be optimal. Algorithm results are shown in bold if they equal the BKS. *AWA17 used an iteration number as stopping criterion and exceeded the allowed time limit in several *medium* instances. **JAR19 used an iteration number as stopping criterion and did not report the used computation time. ***ABU21 reported using relaxed timeout conditions of 5000 seconds for the *medium* track.

Instance	BKS	BUR14	VAL12	LÜ12	TAS15	AWA15	AWA17*	ZHE17	JAR19**	ABD21	ABU21***
medium early 01	**240**	244	**240**	**240**	**240**	245	243	**240**	241	243	**240**
medium early 02	**240**	241	**240**	**240**	**240**	245	242	**240**	241	241	**240**
medium early 03	**236**	238	**236**	**236**	**236**	242	238	**236**	**236**	238	**236**
medium early 04	**237**	240	**237**	**237**	238	240	240	**237**	**237**	240	**237**
medium early 05	**303**	308	**303**	**303**	**303**	308	305	**303**	**303**	**303**	**303**
medium late 01	**157**	187	159	164	161	174	169	158	161	**157**	161
medium late 02	**18**	22	20	20	19	31	26	20	**18**	31	**18**
medium late 03	**29**	46	30	30	30	38	34	30	**29**	**29**	**29**
medium late 04	**35**	49	36	**35**	**35**	48	42	**35**	**35**	**35**	**35**
medium late 05	**107**	161	113	117	112	134	131	111	**107**	140	**107**
medium hidden 01	**111**		131	122	122	155	143	117	**111**	**111**	**111**
medium hidden 02	**219**		221	224	221	254	248	225	220	**219**	220
medium hidden 03	**34**		38	35	**34**	54	49	**34**	**34**	**34**	**34**
medium hidden 04	**78**		81	80	79	94	87	79	**78**	**78**	**78**
medium hidden 05	**118**		122	120	124	177	169	122	119	**118**	119

Table A.4: Best obtained results to the *long* track of the INRC-I benchmark. Best known solutions (BKS) are shown in bold if they are known to be optimal. Algorithm results are shown in bold if they equal the BKS. *JAR19 used an iteration number as stopping criterion and did not report the used computation time. **ABU21 reported using relaxed timeout conditions of 20 hours for the *long* track.

Instance	BKS	BUR14	VAL12	LÜ12	TAS15	AWA15	AWA17	ZHE17	JAR19*	ABD21	ABU21**
long early 01	**197**	198	**197**	**197**	**197**	**197**	**197**	**197**	198	**197**	**197**
long early 02	**219**	223	**219**	222	**219**	229	226	**219**	220	221	**219**
long early 03	**240**	242	**240**	**240**	**240**	**240**	**240**	**240**	**240**	**240**	**240**
long early 04	**303**	305	**303**	**303**	**303**	**303**	**303**	**303**	**303**	**303**	**303**
long early 05	**284**	286	**284**	**284**	**284**	**284**	**284**	**284**	**284**	**284**	**284**
long late 01	**235**	286	239	237	239	257	253	**235**	**235**	240	**235**
long late 02	**229**	290	231	**229**	234	263	256	**229**	**229**	**229**	**229**
long late 03	**220**	290	222	222	227	262	256	221	**220**	**220**	**220**
long late 04	**221**	280	228	227	232	261	263	224	**221**	**221**	**221**
long late 05	**83**	110	**83**	**83**	**83**	102	98	**83**	**83**	89	**83**
long hidden 01	**346**		363	**346**	349	400	380	349	**346**	**346**	**346**
long hidden 02	**89**		106	**89**	**89**	117	110	**89**	**89**	**89**	**89**
long hidden 03	**38**		**38**	**38**	**38**	51	44	**38**	**38**	**38**	**38**
long hidden 04	**22**		**22**	**22**	**22**	29	27	**22**	**22**	**22**	**22**
long hidden 05	**41**		**41**	45	**41**	56	53	**41**	**41**	**41**	**41**

Table A.5: Summarized results for the INRC-I benchmark. *(AWA17) used an iteration number as stopping criterion and exceeded the allowed time limit in several *sprint* and *medium* instances. **(JAR19) used an iteration number as stopping criterion and did not report the used computation time. ***(ABU21) reported using relaxed timeout conditions of 1000 seconds, 5000 seconds and 20 hours for the *sprint*, *medium* and *long* tracks, respectively.

	BUR14	VAL12	LÜ12	TAS15	AWA15	AWA17*	ZHE17	JAR19**	ABD21	ABU21***
Sprint track										
Instances tested	20	30	30	30	30	30	25	30	30	30
BKS found	19	21	29	29	29	29	24	27	23	29
Ratio BKS found	0.95	0.70	0.97	0.97	0.97	0.97	0.96	0.90	0.77	0.97
Medium track										
Instances tested	10	15	15	15	15	15	15	15	15	15
BKS found	0	5	6	7	0	0	7	10	9	12
Ratio BKS found	0.00	0.33	0.40	0.47	0.00	0.00	0.47	0.67	0.60	0.80
Long track										
Instances tested	10	15	15	15	15	15	15	15	15	15
BKS found	0	9	10	10	4	4	12	13	12	15
Ratio BKS found	0.00	0.60	0.67	0.67	0.27	0.27	0.80	0.87	0.80	1.00
All tracks										
Instances tested	40	60	60	60	60	60	55	60	60	60
BKS found	19	35	45	46	33	33	43	50	44	56
Ratio BKS found	0.48	0.58	0.75	0.77	0.55	0.55	0.78	0.83	0.73	0.93

Table A.6: Problem size indicators of the Shift Scheduling benchmark instances

Instance	**Planning period**	**Employees**	**Shift types**
1	2 weeks	8	1
2	2 weeks	14	2
3	2 weeks	20	3
4	4 weeks	10	2
5	4 weeks	16	2
6	4 weeks	18	3
7	4 weeks	20	3
8	4 weeks	30	4
9	4 weeks	36	4
10	4 weeks	40	5
11	4 weeks	50	6
12	4 weeks	60	10
13	4 weeks	120	18
14	6 weeks	32	4
15	6 weeks	45	6
16	8 weeks	20	3
17	8 weeks	30	4
18	12 weeks	22	3
19	12 weeks	40	5
20	26 weeks	50	6
21	26 weeks	100	8
22	52 weeks	50	10
23	52 weeks	100	16
24	52 weeks	150	32

Table A.7: Best obtained results for the Shift Scheduling benchmark instances with a time limit of 60 minutes. Best known solutions (BKS) are shown in bold if they are known to be optimal. Algorithm results are shown in bold if they equal the BKS. If no algorithm listed here found the BKS, the best solution among them is underlined. Dash symbols (-) indicate no feasible solution was found within the time limit. *This result may be incorrect, as Strandmark et al., 2020 reported having found a lower bound higher than this. Therefore, the BKS shown for this instance does not take this result into account.

Instance	BKS	BUR14	RAH17	TUR20	ABD21b	ABD21	GOH22	CHE23	Gurobi 5.6.3
1	**607**	**607**	**607**	**607**	**607**	**607**	**607**		**607**
2	**828**	837	**828**	**828**	835	**828**	**828**		**828**
3	**1001**	1003	**1001**	**1001**	1012	**1001**	**1001**		**1001**
4	**1716**	1718	**1716**	**1716**	1728	**1716**	**1716**		**1716**
5	**1143**	1358	**1143**	**1143**	1257	1237	1150	1145	**1143**
6	**1950**	2258	**1950**	**1950**	2167	2141	2048	**1950**	**1950**
7	**1056**	1269	**1056**	**1056**	1110	1080	1077	1082	**1056**
8	**1300**	2260	1344	1322	1443	1452	1374	1322	1323
9	**439**	463	**439**	**439**	456	446	491	440	**439**
10	**4631**	4797	**4631**	**4631**	4784	4656	4663	**4631**	**4631**
11	**3443**	3661	**3443**	**3443**	3661	3512	3457	**3443**	**3443**
12	**4040**	5211	**4040**	**4040**	4344	4119	4173	4153	**4040**
13	**1348**	3037	1905	2900	2712	2120	3224	2769	3109
14	**1278**	1847	1279	1280	1465	1344	1324	1297	1280
15	**3829**	5935	3928	4190	4838	4637	4366	5920	4964
16	**3225**	4048	**3225**	**3225**	3981	3458	3435	3351	3233
17	**5746**	7835	5750	5848	6420	6190	5913	5748	5851
18	**4459**	6404	4662	4650	5526	5095	4904	4954	4760
19	**3149**	5531	3224	3218	5670	4281	3425	4338	5420
20	**4769**	9750	4913		18876	7274	5063	5905	-
21	**21133**	36688	23191		58995	26263	21731	22282	-
22	**30241**	516686	32126		142778	56091	34855	53546	-
23	17428	54384	3794*		206744	51699	18947	38752	-
24	42463	156858	2281440		792331	226490	56001	402049	-

Table A.8: Best obtained results for the Shift Scheduling benchmark instances with a time limit of 10 minutes. Best known solutions (BKS) are shown in bold if they are known to be optimal. Algorithm results are shown in bold if they equal the BKS. If for an instance no algorithm listed here found the BKS, the best solution among them is underlined. Dash symbols (-) indicate no feasible solution was found within the time limit.

Instance	BKS	BUR14	RAH17	TUR20	GOH22	Gurobi 5.6.3
1	**607**	**607**	**607**	**607**	**607**	**607**
2	**828**	923	**828**	**828**	**828**	**828**
3	**1001**	1003	**1001**	**1001**	1002	**1001**
4	**1716**	1719	**1716**	**1716**	**1716**	**1716**
5	**1143**	1439	**1143**	**1143**	1155	**1143**
6	**1950**	2344	**1950**	**1950**	2054	**1950**
7	**1056**	1284	**1056**	**1056**	1080	**1056**
8	**1300**	2529	1364	1341	1374	8995
9	**439**	474	**439**	**439**	488	**439**
10	**4631**	4999	**4631**	**4631**	4664	**4631**
11	**3443**	3967	**3443**	**3443**	3459	**3443**
12	**4040**	5611	4042	4044	4161	4045
13	**1348**	8707	3109	3200	3201	500410
14	**1278**	2542	1281	1295	1333	1482
15	**3829**	6049	4144	4420	4490	78144
16	**3225**	4343	3306	3253	3453	3521
17	**5746**	7835	5760	6138	6012	6149
18	**4459**	6404	5049	5000	4958	7950
19	**3149**	6522	3974	3809	3635	29968
20	**4769**	23531	5242		5137	-
21	**21133**	38294	26977		21833	-
22	**30241**	-	130107		37227	-
23	17428	-	40543		19926	-
24	42463	-	2829680		62387	-

B Rules from the Dutch Working Hours Act

The following rules are obtained from the Dutch Working Hours Act (Arbeidstijdenwet, 2022, §5.2). Only those rules that are relevant for the problem formulation are considered. Article numbers are added in brackets.

- **Daily rest (5:3):** in every 24-hour period, an employee must have a minimum of 11 consecutive hours of rest. An exception to this rule can be made once every 7 days, where the minimum is lowered to 8 hours.
- **Weekly rest (5:5):** an employee must have a minimum consecutive rest time of either
  - 36 hours every 7-day period, or
  - 72 hours every 14-day period. In this case, the rest time may be split into two periods of at least 32 hours each.
- **Working Sundays (5:6.3):** in every 52-week period, an employee must have a minimum of 13 non-working Sundays.
- **Night shifts (5:8):** a working shift is considered a 'night shift' if at least 1 hour is worked between the hours of midnight and 6 am. The following rules apply specifically to night shifts:
  1. A night shift may have a duration of at most 10 hours. (5:8.1)
  2. In every 16-week period, where an employee works at least 16 night shifts, a maximum of 40 working hours per week is allowed. (5:8.2)
  3. An employee must have a minimum of 14 consecutive hours of rest, following a night shift that ends after 2 am. An exception can me made to this rule once every 7 days, where the minimum is lowered to 8 hours. (5:8.4)
  4. Deviating from the first and third rule, at most 5 times in every 14-day period and at most 22 times in every 52-week period, an employee may (5:8.3):
     - work at most 12 hours per night shift,
     - have a rest time of at least 12 hours after such a night shift.
  5. Following a sequence of at least 3 consecutive night shifts, an employee must have a minimum rest time of 46 consecutive hours. (5:8.5)
  6. If a sequence of consecutive shifts for an employee contains at least one night shift, then that sequence may contain at most 7 shifts. This limit may be deviated from in a collective labor agreement, given that the length of such a sequence does not exceed 8 shifts. (5:8.6,5:8.7)
  7. An employee is allowed to work at most (5:8.9):
     - 140 night shifts that end after 2 am in every 52-week period, or
     - 38 hours between the hours of midnight and 6 am in every 2-week period
  8. An employee is allowed to work at most 36 night shifts that end after 2 am in every 16-week period. This limit may be deviated from in a collective labor agreement, given that the seventh rule is respected. (5:8.8)

C Notation

Table C.1: Notation for the sets and parameters used in the descriptions of the soft constraint penalties in Section 4.3 and in the mixed-integer program formulation of the simplified model, described in Section 6.4. *Note that β_s is an indicator for three shift type properties at once: (1) if it is a night shift by the definition given in Section 4.2, (2) if its end time is after 2 am, and (3) if it ends on the day after the day on which it starts. However, in our problem instances, each shift type has either all three or none of these properties, and thus, one combined indicator is used here.

Sets and indices	
E	Set of employees ($e \in E$)
D	Set of days ($d \in D$)
S	Set of shift types ($s \in S$)
S^+	Set of shift types, with day-off assignment o included ($S^+ = S \cup \{o\}$)
Parameters	
$a_{d,s}$	Coverage requirement on day d for shift type s in number of employees
b_s	Priority shift type indicator: 1 if shift type s is a priority shift type, 0 otherwise
c_e^{con}	Contractual workload for employee e in hours
c_e^{max}	Maximum workload for employee e in hours: $c_e^{max} = c_e^{con} + 10$
f_s	Duration of shift type s in hours
$g_{e,d}^{on}$	Day-on request indicator: 1 if employee e has requested to be assigned to a shift on day d, 0 otherwise
$g_{e,d}^{off}$	Day-off request indicator: 1 if employee e has requested a day off on day d, 0 otherwise
$h_{e,d,s}^{on}$	Shift-on request indicator: 1 if employee e has requested to be assigned to shift type s on day d, 0 otherwise
$h_{e,d,s}^{off}$	Shift-off request indicator: 1 if employee e has requested not to be assigned to shift type s on day d, 0 otherwise
j_e	Weight for the requests and preferred shift and day-off sequence lengths of employee e
$k_{e,s}$	Valid shift indicator: 1 if employee e has the required skill to work shift type s, 0 otherwise
$m_{e,d,s}$	Fixed assignment indicator: 1 if employee e has a fixed assignment of shift type s on day d, 0 otherwise
$n_{s,t}$	11 hrs rest indicator: 1 if there is at least 11 hrs rest between shift type s on day d, and shift type t on day $d+1$
$p_{s,t}$	8 hrs rest indicator: 1 if there is at least 8 hrs rest between shift type s on day d, and shift type t on day $d+1$
q_d	Sunday indicator: 1 if day d is a Sunday, 0 otherwise
r_e	Maximum number of Sundays that employee e can work in this period
β_s	Night shift indicator: 1 if shift type s is a night shift, 0 otherwise*
γ_e	Maximum number of night shifts that employee e can work in this period

Table C.2: Notation for the decision and auxiliary variables used in the descriptions of the soft constraint penalties in Section 4.3 and in the mixed-integer program formulation of the simplified model, described in Section 6.4

Decision variables	
$x_{e,d,s}$	Working shift indicator: 1 if employee e works shift type s starting on day d, 0 otherwise
Auxiliary variables	
$y_{d,s}$	Coverage shortage of shift type $s \in S$ on day $d \in D$: $y_{d,s} = \max\{a_{d,s} - \sum_{e \in E} x_{e,d,s}, 0\}$
y_d	Total coverage shortage on day $d \in D$: $y_d = \sum_{s \in S} y_{d,s}$
z_d	Coverage shortage on day d minus one: $z_d = \max\{\big(\sum_{s \in S} y_{d,s}\big) - 1, 0\}$
u_e	Worked overtime hours for employee e: $u_e = \max\{\big(\sum_{d \in D} \sum_{s \in S} x_{e,d,s} f_s\big) - c_e^{con}, 0\}$
$v_{e,d}$	Daily rest exception indicator: 1 if the daily rest exception is used on day d for employee e, 0 otherwise
$w_{e,d}$	Worked Sunday indicator: 1 if employee e works on day d which is a Sunday, 0 otherwise
ζ_e^{on}	Number of sequences of consecutive shifts that employee e works and are of the preferred length
ζ_e^{off}	Number of sequences of consecutive days off for employee e that are of the preferred length
η_e^{on}	Total number of sequences of consecutive shifts that employee e works
η_e^{off}	Total number of sequences of consecutive days off for employee e

D Additional results

In this appendix, the results figures and tables are shown, that were left out of Chapter 7 for readibility.

D.1. Construction methods results

Table D.1: Results of both construction methods on all nine instances, using the sorting criteria shown in Table 7.1. The mean, standard deviation (SD), minimum and maximum are shown of the total roster penalties over 25 runs. The values are rounded to integers for clarity.

	Construction-per-shift				**Construction-per-employee**			
Instance	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**
O4	85,109	1,360	83,237	88,537	216,842	3,708	209,693	224,220
O5	122,446	2,471	118,094	129,154	301,589	5,136	291,735	313,082
O6	103,879	1,381	101,862	106,562	249,874	7,719	231,498	264,486
T4	7,945	323	7,402	8,737	20,664	1,487	18,333	24,593
T5	8,730	315	8,299	9,354	23,812	1,496	21,026	26,876
T6	13,551	601	12,429	14,625	36,914	1,376	34,373	39,782
V4	8,818	67	8,695	8,925	10,443	167	10,127	10,740
V5	6,672	104	6,508	6,883	7,810	230	7,377	8,244
V6	19,417	670	18,157	20,920	29,275	850	27,394	30,954

Table D.2: Results of both construction methods on all nine instances, using the sorting criteria shown in Table 7.2. The mean, standard deviation (SD), minimum and maximum are shown of the total roster penalties over 25 runs. The values are rounded to integers for clarity.

	Construction-per-shift				**Construction-per-employee**			
Instance	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**
O4	76,812	1,823	73,442	80,287	215,408	3,859	208,932	222,033
O5	110,178	2,517	106,452	115,643	301,224	4,508	293,183	308,840
O6	93,302	2,167	89,553	98,335	248,571	5,187	237,400	259,389
T4	7,815	383	7,186	8,420	20,436	1,392	16,965	23,190
T5	8,708	299	7,979	9,266	23,192	1,377	20,270	26,181
T6	13,183	468	12,481	14,238	36,518	1,311	34,104	39,287
V4	8,493	69	8,409	8,664	10,447	158	10,173	10,733
V5	6,414	82	6,246	6,565	7,806	239	7,474	8,523
V6	19,672	755	18,209	20,922	29,370	883	27,313	30,861

Table D.3: Results of Construction-per-shift (CPS) and Construction-per-employee (CPE), using the sorting criteria shown in Table 7.1, broken down into the different penalty categories. The values are averaged over 25 runs and rounded to integers for clarity. The abbreviations of the different penalty categories are clarified in Table D.7.

	TP		**CP**		**CSP**		**OHSP**		**RP**		**PSSLP**		**PDOSLP**	
Instance	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**
O4	85,109	216,842	29,768	20,864	49,316	189,744	2,531	2,617	1,124	1,106	1,343	1,356	1,027	1,155
O5	122,446	301,589	35,212	26,912	82,128	269,228	1,642	1,658	669	756	1,590	1,555	1,204	1,481
O6	103,879	249,874	32,512	24,080	66,416	220,756	2,413	2,428	563	517	1,147	1,182	827	911
T4	7,945	20,664	3,912	5,940	896	11,568	1,114	861	350	438	975	984	698	874
T5	8,730	23,812	5,640	8,060	736	13,248	543	496	197	253	996	1,016	618	740
T6	13,551	36,914	8,480	11,252	2,264	22,812	1,059	831	172	221	1,001	1,044	574	754
V4	8,818	10,443	6,504	6,984	0	400	190	735	374	464	1,017	1,029	733	830
V5	6,672	7,810	4,204	4,392	100	484	116	507	366	468	1,098	1,103	788	857
V6	19,417	29,275	12,536	13,580	3,876	12,352	725	849	308	360	1,186	1,157	786	977

Table D.4: Run times and roster properties related to the different soft constraint penalties for Construction-per-shift (CPS) and Construction-per-employee (CPE), using the sorting criteria shown in Table 7.1. The values are averaged over 25 runs and rounded somewhat for clarity. The abbreviations for the different roster properties are clarified in Table D.8.

	RT		**US**		**UPS**		**OHPE**		**UR**		**PSSLR**		**PDOSLR**	
Instance	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**
O4	0.68	0.87	178	192	30.0	4.2	3.8	4.0	93.9	97.8	0.12	0.12	0.33	0.26
O5	0.65	0.67	228	245	31.0	6.0	3.0	3.0	58.3	66.3	0.12	0.13	0.32	0.21
O6	0.66	0.71	204	209	30.4	7.9	3.9	3.9	57.4	53.8	0.12	0.11	0.34	0.28
T4	0.14	0.18	27	47	3.0	3.0	4.1	3.4	22.1	26.9	0.16	0.15	0.48	0.33
T5	0.15	0.19	28	53	7.0	7.0	2.6	2.2	10.1	12.9	0.13	0.11	0.49	0.37
T6	0.22	0.15	41	69	11.0	11.0	4.1	3.4	8.7	11.3	0.10	0.06	0.50	0.35
V4	0.12	0.18	13	18	13.0	13.0	0.6	2.3	37.3	40.2	0.10	0.10	0.38	0.32
V5	0.12	0.18	10	12	8.0	8.0	0.4	1.9	58.3	67.8	0.09	0.11	0.37	0.32
V6	0.18	0.16	53	64	18.0	18.0	2.3	2.9	23.2	26.1	0.13	0.14	0.43	0.30

Table D.5: Results of Construction-per-shift (CPS) and Construction-per-employee (CPE), using the sorting criteria shown in Table 7.2, broken down into the different penalty categories. The values are averaged over 25 runs and rounded to integers for clarity. The abbreviations of the different penalty categories are clarified in Table D.7.

	TP		**CP**		**CSP**		**OHSP**		**RP**		**PSSLP**		**PDOSLP**	
Instance	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**
O4	76,812	215,408	18,148	20,840	53,412	188,344	2,533	2,624	345	1,096	1,329	1,352	1,045	1,151
O5	110,178	301,224	22,852	26,932	82,600	268,840	1,644	1,655	274	757	1,589	1,560	1,219	1,480
O6	93,302	248,571	20,640	24,072	68,072	219,416	2,410	2,467	208	516	1,148	1,182	825	918
T4	7,815	20,436	4,012	5,928	920	11,352	1,108	859	84	433	979	983	712	880
T5	8,708	23,192	5,712	8,008	832	12,684	523	496	12	249	998	1,015	631	739
T6	13,183	36,518	8,376	11,200	2,068	22,464	1,087	835	67	221	1,003	1,044	581	754
V4	8,493	10,447	6,520	6,980	12	404	168	734	49	473	1,010	1,030	733	826
V5	6,414	7,806	4,200	4,384	120	472	109	500	103	474	1,090	1,107	792	869
V6	19,672	29,370	12,712	13,592	4,180	12,436	730	849	84	365	1,185	1,159	781	969

Table D.6: Run times and roster properties related to the different soft constraint penalties for Construction-per-shift (CPS) and Construction-per-employee (CPE), using the sorting criteria shown in Table 7.2. The values are averaged over 25 runs and rounded somewhat for clarity. The abbreviations for the different roster properties are clarified in Table D.8.

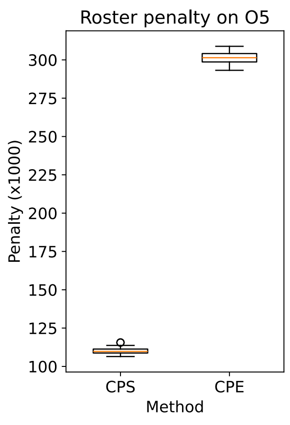
	RT		**US**		**UPS**		**OHPE**		**UR**		**PSSLR**		**PDOSLR**	
Instance	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**	**CPS**	**CPE**
O4	0.54	0.58	181	191	0.0	4.4	3.8	4.0	27.4	97.5	0.13	0.12	0.32	0.26
O5	0.57	0.51	229	245	0.0	6.1	3.0	3.0	24.3	66.2	0.12	0.13	0.32	0.21
O6	0.52	0.52	205	209	0.4	7.9	3.9	4.0	20.0	53.6	0.12	0.11	0.34	0.28
T4	0.10	0.12	28	47	3.0	3.0	4.1	3.4	5.3	26.8	0.16	0.15	0.45	0.33
T5	0.12	0.11	29	52	7.0	7.0	2.5	2.2	0.6	12.7	0.13	0.11	0.48	0.38
T6	0.11	0.12	40	68	11.0	11.0	4.3	3.4	3.6	11.4	0.11	0.06	0.49	0.35
V4	0.09	0.12	13	18	13.0	13.0	0.6	2.3	7.3	41.0	0.11	0.10	0.37	0.33
V5	0.08	0.12	10	12	8.0	8.0	0.4	1.8	12.4	69.0	0.11	0.10	0.37	0.31
V6	0.14	0.12	55	64	18.0	18.0	2.4	2.9	6.4	26.4	0.13	0.14	0.43	0.31

Table D.7: Penalty category abbreviations. The requests penalty is the sum of the day-on, day-off, shift-on and shift-off requests penalties.

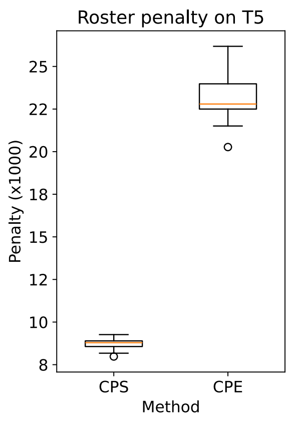
Penalty category	**Abbreviation**
Total penalty	TP
Coverage penalty	CP
Coverage spread penalty	CSP
Overtime hours spread penalty	OHSP
Requests penalty	RP
Preferred shift sequence length penalty	PSSLP
Preferred day-off sequence length penalty	PDOSLP

Table D.8: Run time and roster property abbreviations. The number of unsatisfied requests is the sum of the numbers of unsatisfied day-on, day-off, shift-on and shift-off requests. The preferred shift/day-off sequence length ratio for an employee is equal to the number of shift/day-off sequences of the preferred length, divided by the total number of shift/day-off sequences. These ratios are then averaged over the employees.

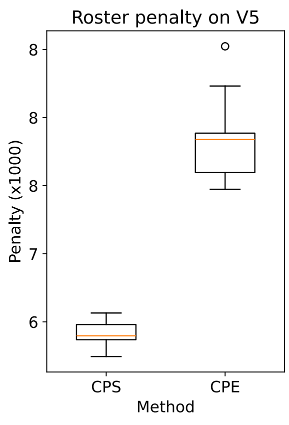
Property	**Abbreviation**
Run time in seconds	RT
Number of unplanned shifts	US
Number of unplanned priority shifts	UPS
Average worked overtime hours per employee	OHPE
Number of unsatisfied requests	UR
Average preferred shift sequence length ratio	PSSLR
Average preferred day-off sequence length ratio	PDOSLR



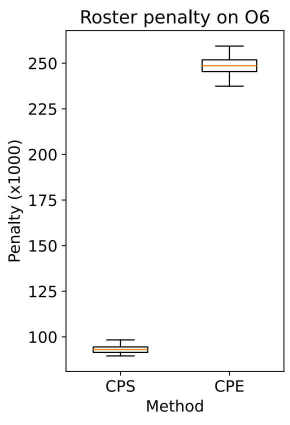
(a)



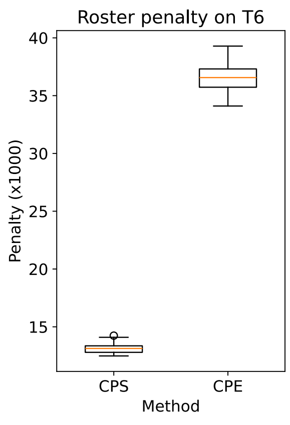
(b)



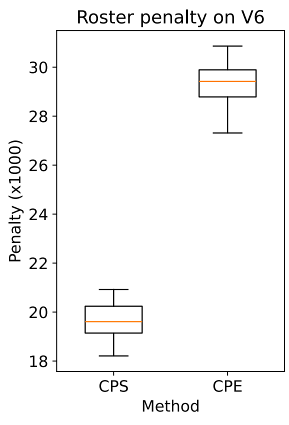
(c)



(d)



(e)



(f)

Figure D.1: Box plots of the results of Construction-per-shift (CPS) and Construction-per-employee (CPE), using the sorting criteria shown in Table 7.2, on the May and June instances over 25 runs per instance.

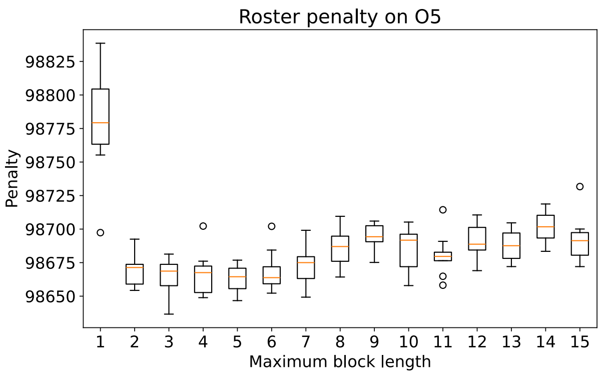
D.2. Overall frameworks results

Table D.9: Fractions of the total number of improving moves (IMF) of neighborhood structures (NS) for Variable Neighborhood Search with a maximum block length x of 19. Note that Hx neighborhood structures with block lengths x greater than 15 were not added, because in a scheduling period of a month there are no two non-overlapping blocks of 16 or more days that can be swapped for an employee.

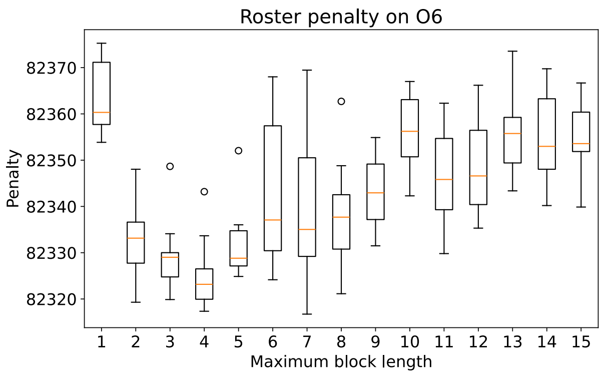
NS	IMF	NS	IMF	NS	IMF
UC1	0.02380	V1	0.28258	C1	0.02710
UC2	0.37228	V2	0.06677	C2	0.00090
H1	0.17608	V3	0.02008	C3	0.00027
H2	0.00376	V4	0.00831	C4	0.00017
H3	0.00093	V5	0.00416	C5	0.00006
H4	0.00022	V6	0.00237	C6	0.00006
H5	0.00014	V7	0.00155	C7	0.00002
H6	0.00005	V8	0.00133	C8	0.00000
H7	0.00003	V9	0.00114	C9	0.00000
H8	0.00000	V10	0.00088	C10	0.00000
H9	0.00000	V11	0.00079	C11	0.00000
H10	0.00001	V12	0.00075	C12	0.00000
H11	0.00000	V13	0.00063	C13	0.00000
H12	0.00000	V14	0.00059	C14	0.00000
H13	0.00000	V15	0.00046	C15	0.00000
H14	0.00000	V16	0.00042	C16	0.00000
H15	0.00000	V17	0.00043	C17	0.00000
		V18	0.00046	C18	0.00000
		V19	0.00042	C19	0.00000

Table D.10: Results of the total penalty of Variable Neighborhood Search with different orderings, on all nine instances, averaged over 10 runs per instance. The tested orderings are displayed in Table 7.4.

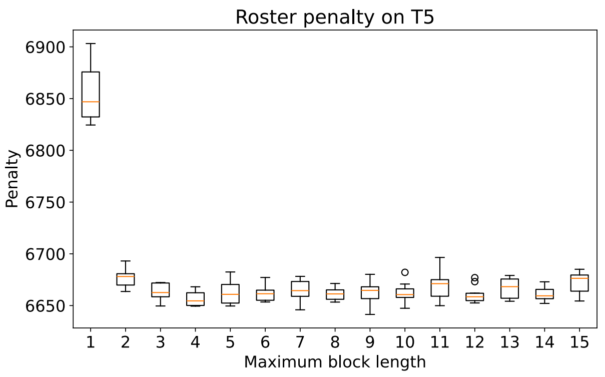
Instance	O	IBL	WNR	UCFL
O4	64,859.19	64,854.46	64,865.46	65,119.93
O5	98,646.05	98,642.78	98,644.89	98,699.62
O6	82,307.28	82,312.50	82,313.93	82,324.26
T4	5,871.50	5,867.51	5,869.37	5,879.62
T5	6,658.26	6,660.05	6,662.52	6,673.99
T6	10,670.41	10,665.15	10,668.74	10,675.01
V4	7,508.14	7,507.69	7,510.11	7,507.39
V5	5,313.94	5,316.71	5,321.42	5,320.49
V6	14,031.96	14,040.28	14,061.69	14,100.37



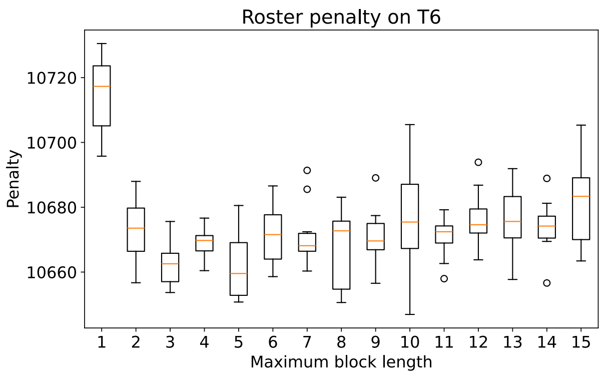
(a)



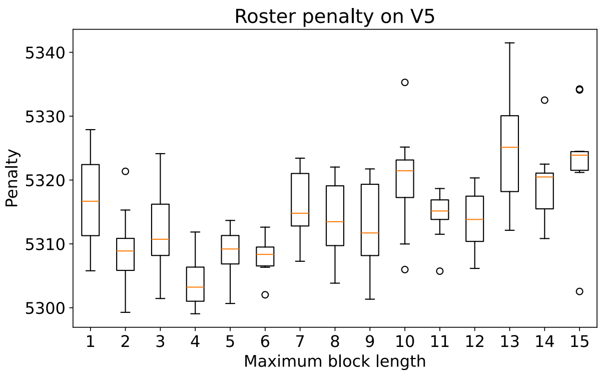
(b)



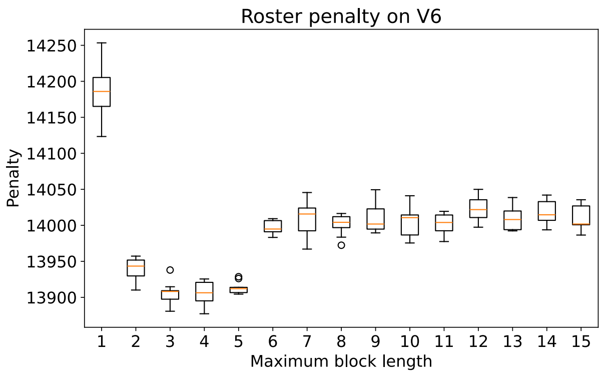
(c)



(d)

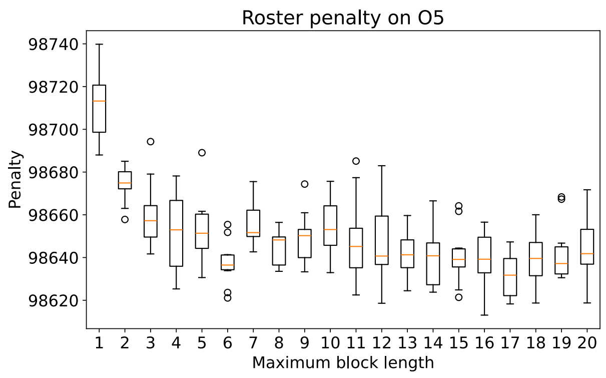


(e)

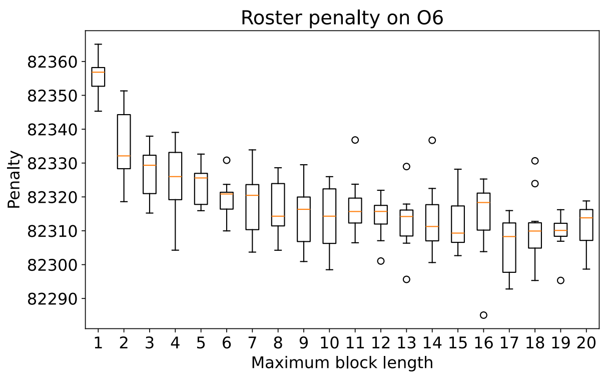


(f)

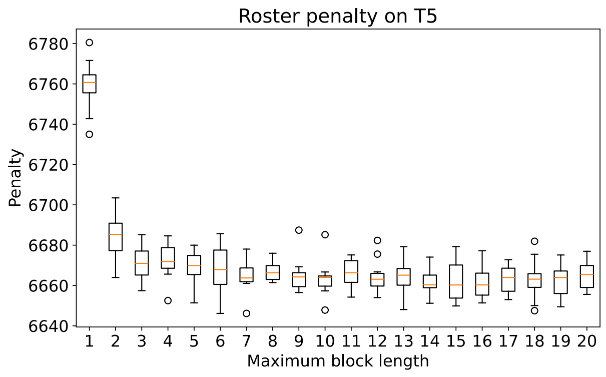
Figure D.2: Box plots of the results of Simulated Annealing with different maximum block lengths on the May and June instances over 10 runs per instance.



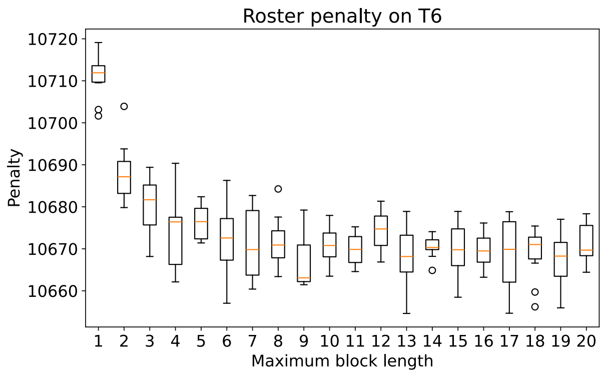
(a)



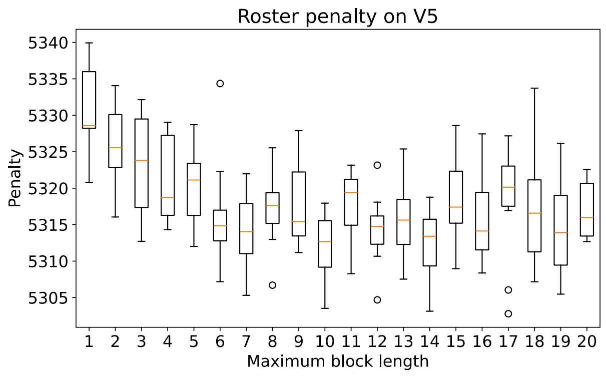
(b)



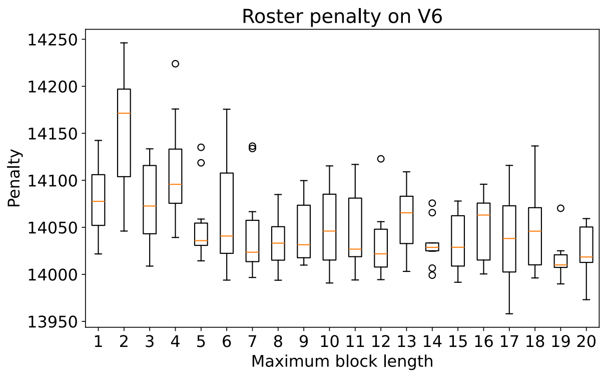
(c)



(d)

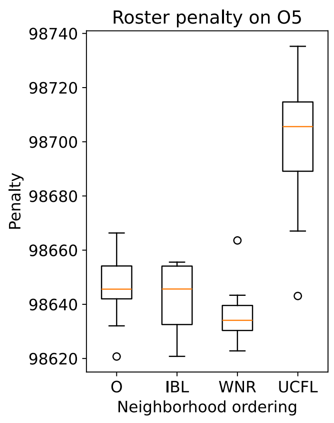


(e)

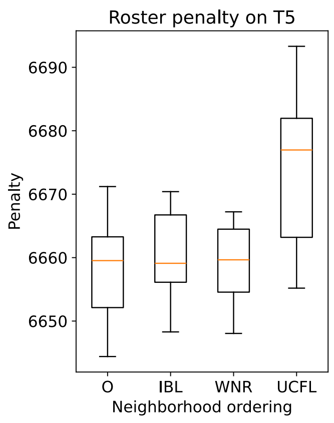


(f)

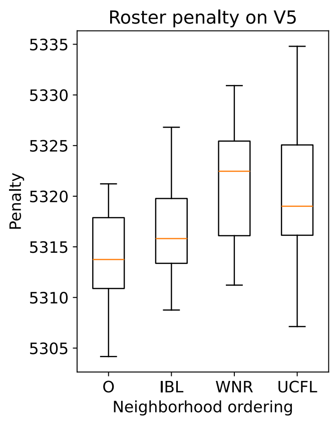
Figure D.3: Box plots of the results of Variable Neighborhood Search with different maximum block lengths on the May and June instances over 10 runs per instance.



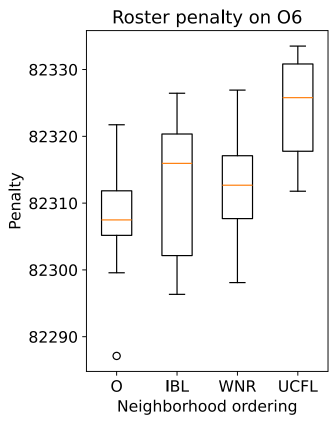
(a)



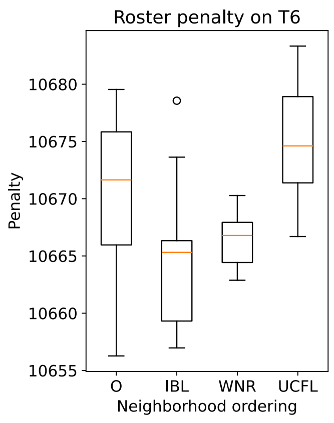
(b)



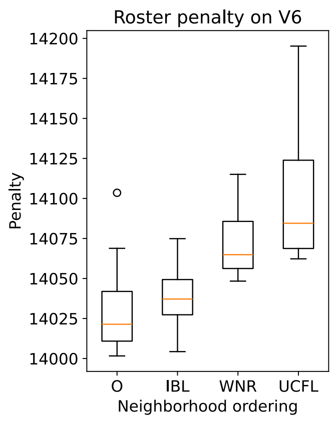
(c)



(d)

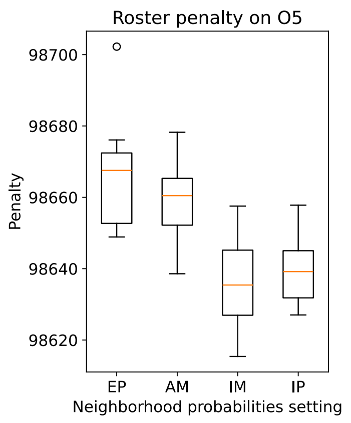


(e)

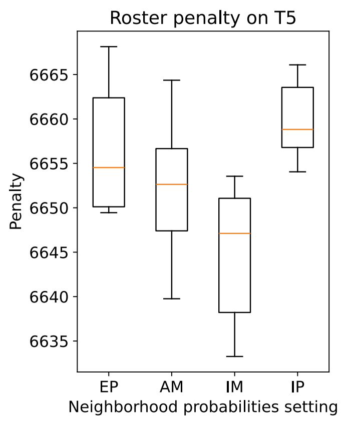


(f)

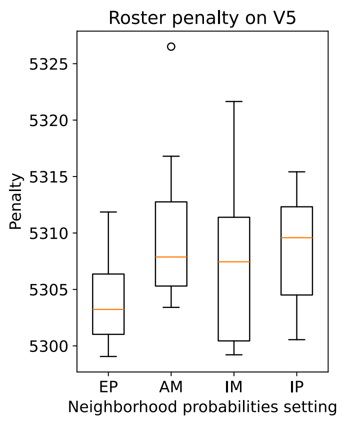
Figure D.4: Box plots of the results of Variable Neighborhood Search with different neighborhood orderings on the May and June instances over 10 runs per instance. The tested orderings are displayed in Table 7.4.



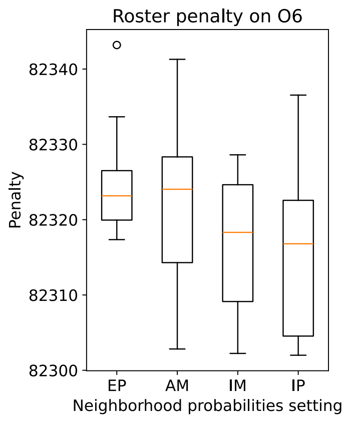
(a)



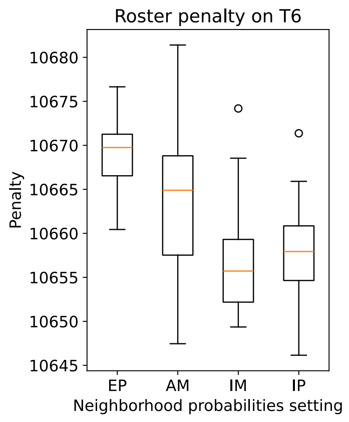
(b)



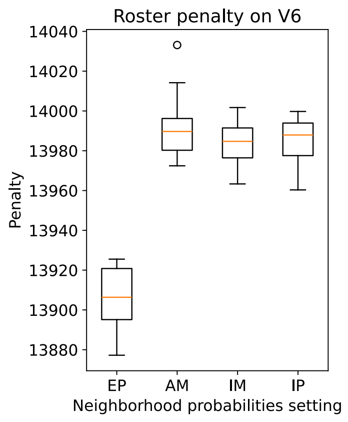
(c)



(d)

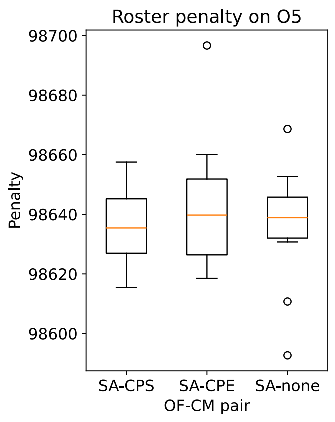


(e)

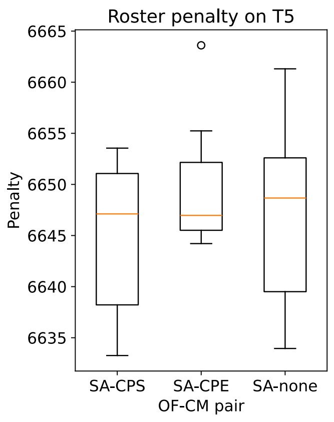


(f)

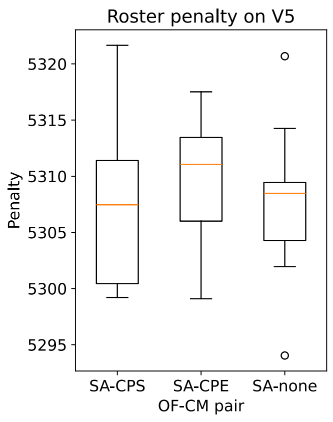
Figure D.5: Box plots of the results of Simulated Annealing with different neighborhood probability distributions on the May and June instances over 10 runs per instance. The shown settings are equal probabilities (EP) and probabilities by accepted moves (AM), improving moves (IM) and improved penalty (IP).



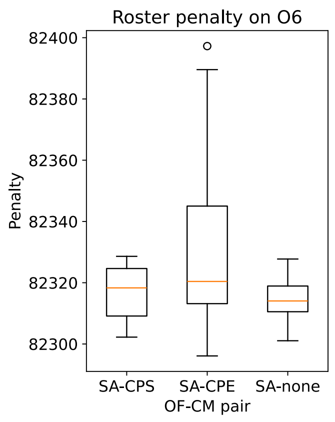
(a)



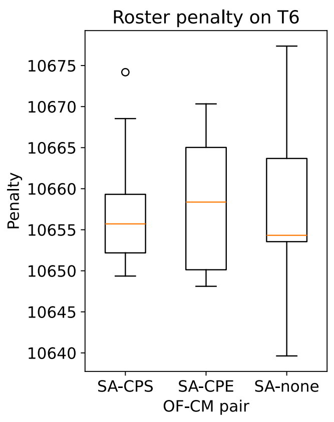
(b)



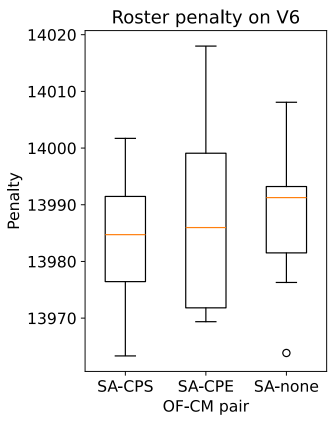
(c)



(d)

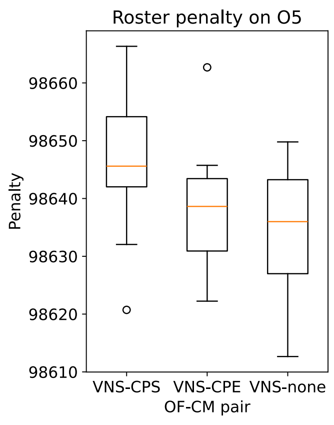


(e)

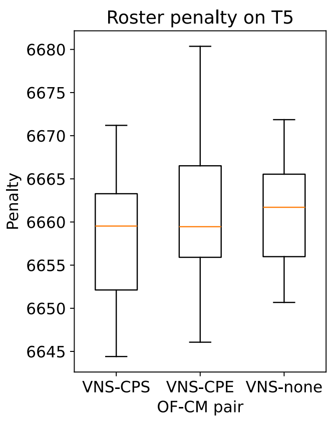


(f)

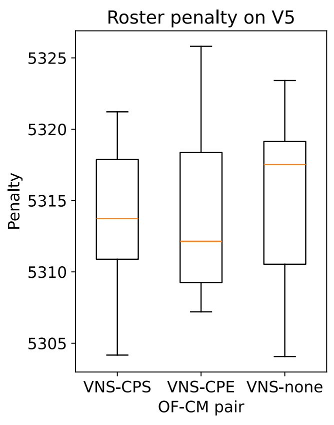
Figure D.6: Box plots of the results of different pairs of overall framework (OF) Simulated Annealing (SA) and construction methods (CM) on the May and June instances over 10 runs per instance.



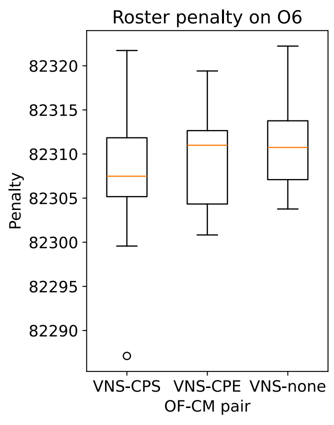
(a)



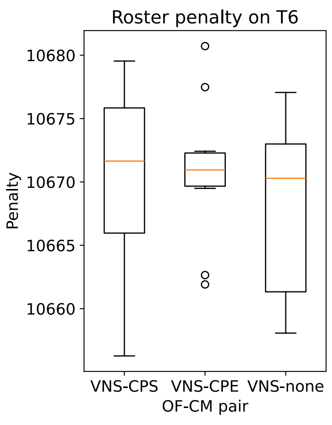
(b)



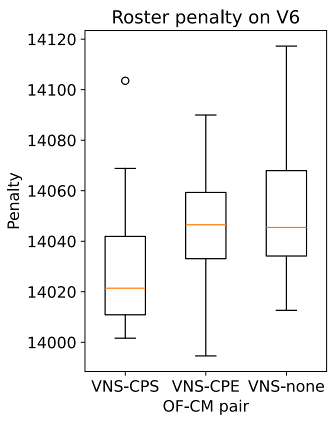
(c)



(d)

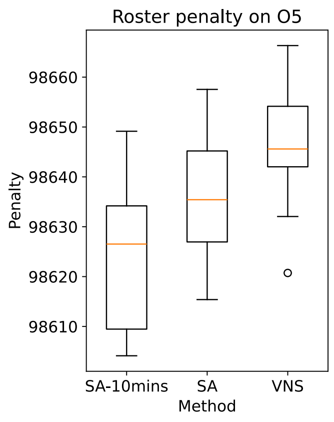


(e)

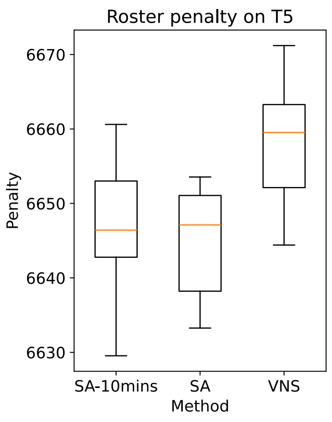


(f)

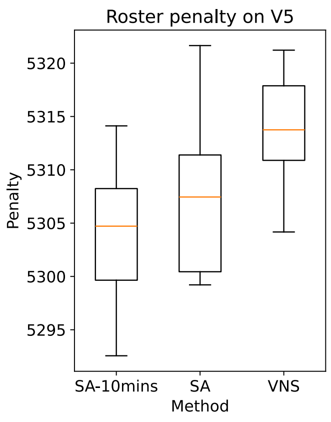
Figure D.7: Box plots of the results of different pairs of overall framework (OF) Variable Neighborhood Search (VNS) and construction methods (CM) on the May and June instances over 10 runs per instance.



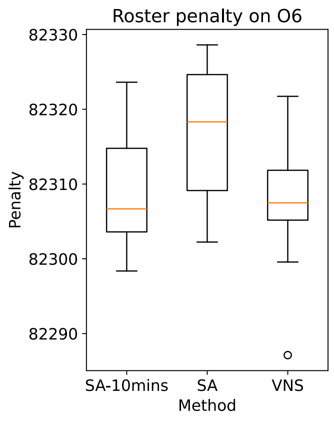
(a)



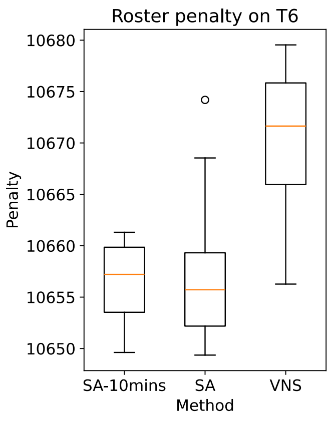
(b)



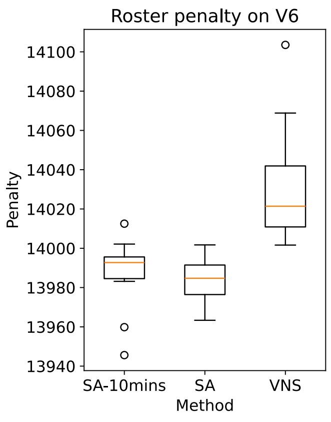
(c)



(d)

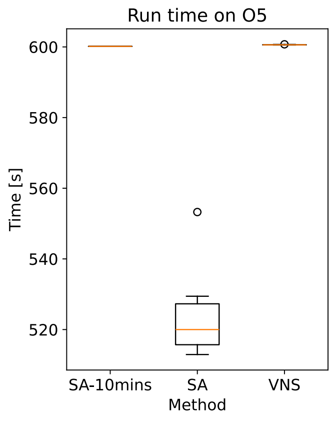


(e)

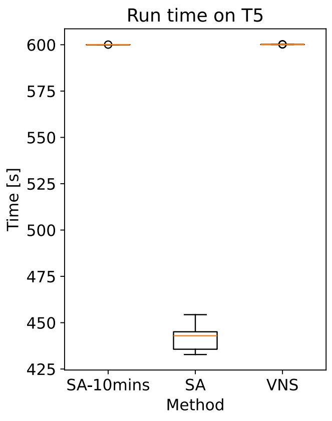


(f)

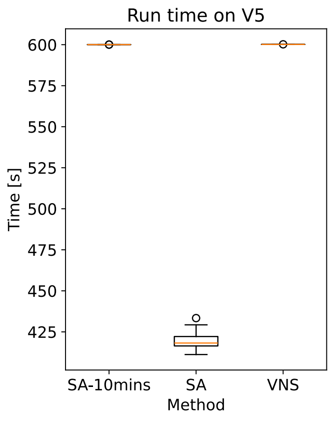
Figure D.8: Box plots of the results of Simulated Annealing, with and without the minimum temperature stopping criterion (SA and SA-10mins, respectively), and Variable Neighborhood Search (VNS) on the April instances over 10 runs per instance. Note that SA-10mins and VNS always reach the time limit of 10 minutes, whereas SA can be stopped earlier by the minimum temperature stopping conditions, see Figures 7.7 and D.9.



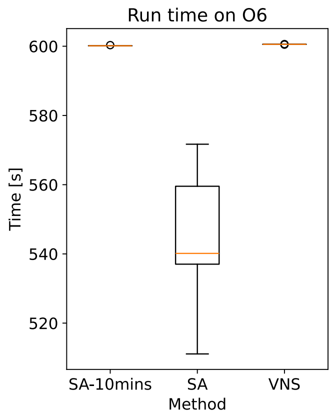
(a)



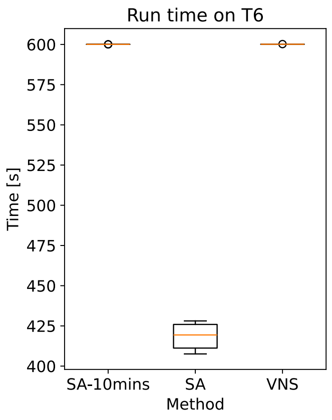
(b)



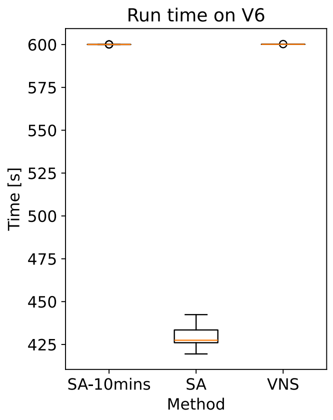
(c)



(d)



(e)



(f)

Figure D.9: Box plots of the run times of Simulated Annealing, with and without the temperature stopping criterion (SA and SA-10mins, respectively), and Variable Neighborhood Search (VNS) on the April instances over 10 runs per instance.

Table D.11: Results of Simulated Annealing without temperature stopping criterion (SA) and Variable Neighborhood Search (VNS) on all nine instances. The mean, standard deviation (SD), minimum and maximum are shown of the total roster penalties over 10 runs. The values are rounded somewhat for clarity.

	Simulated Annealing				**Variable Neighborhood Search**			
Instance	**Mean**	**SD**	**Min**	**Max**	**Mean**	**SD**	**Min**	**Max**
O4	64,854	11.3	64,830	64,869	64,859	16.4	64,839	64,889
O5	98,624	15.1	98,604	98,649	98,646	13.0	98,621	98,666
O6	82,309	7.6	82,298	82,324	82,307	9.5	82,287	82,322
T4	5,862	7.1	5,848	5,874	5,872	8.9	5,860	5,887
T5	6,647	8.8	6,630	6,661	6,658	8.0	6,644	6,671
T6	10,656	4.1	10,650	10,661	10,670	7.7	10,656	10,680
V4	7,501	7.0	7,484	7,510	7,508	2.9	7,501	7,511
V5	5,304	6.4	5,293	5,314	5,314	5.4	5,304	5,321
V6	13,987	19.8	13,946	14,013	14,032	32.4	14,002	14,104

Table D.12: Roster properties related to the different soft constraint penalties after application of Simulated Annealing without temperature stopping criterion (SA) and Variable Neighborhood Search (VNS). The values are averaged over 10 runs and rounded somewhat for clarity. The abbreviations for the different roster properties are clarified in Table D.8.

	US		**UPS**		**OHPE**		**UR**		**PSSLR**		**PDOSLR**	
Instance	**SA**	**VNS**	**SA**	**VNS**	**SA**	**VNS**	**SA**	**VNS**	**SA**	**VNS**	**SA**	**VNS**
O4	177	177	0	0	3.38	3.40	16.0	16.0	0.53	0.53	0.71	0.71
O5	228	228	0	0	2.69	2.71	20.0	20.0	0.54	0.54	0.71	0.71
O6	203	203	0	0	3.38	3.39	17.0	17.0	0.52	0.52	0.74	0.74
T4	24	24	3	3	4.49	4.49	0.6	0.5	0.54	0.53	0.64	0.64
T5	26	26	7	7	1.99	1.99	0.0	0.0	0.56	0.54	0.70	0.70
T6	36	36	11	11	4.67	4.67	0.0	0.0	0.55	0.54	0.71	0.71
V4	13	13	13	13	0.17	0.17	6.0	6.0	0.63	0.61	0.66	0.66
V5	10	10	8	8	0.36	0.36	10.0	10.0	0.58	0.57	0.67	0.67
V6	40	40	18	18	1.50	1.57	10.2	10.7	0.52	0.52	0.69	0.69

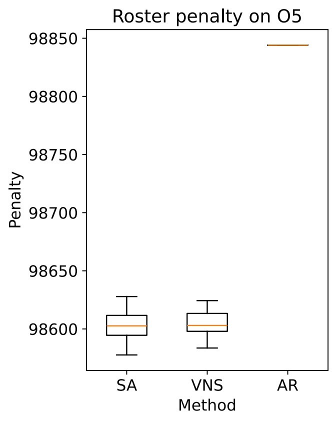
Table D.13: Comparison of the results of Construction-per-shift (CPS) and Simulated Annealing (SA) on all nine instances, averaged over 25 runs for CPS and over 10 runs for SA. The numbers are rounded to integers for clarity. Abbreviations of the penalty categories are shown in Table D.7.

	TP		**CP**		**CSP**		**OHSP**		**RP**		**PSSLP**		**PDOSLP**	
Instance	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**
O4	76,812	64,854	18,148	17,700	53,412	43,500	2,533	2,205	345	171	1,329	747	1,045	531
O5	110,178	98,624	22,852	22,800	82,600	72,800	1,644	1,492	274	220	1,589	787	1,219	525
O6	93,302	82,309	20,640	20,300	68,072	58,800	2,410	2,077	208	178	1,148	589	825	365
T4	7,815	5,862	4,012	3,600	920	0	1,108	1,205	84	7	979	562	712	488
T5	8,708	6,647	5,712	5,400	832	0	523	319	12	0	998	532	631	396
T6	13,183	10,656	8,376	8,000	2,068	600	1,087	1,186	67	0	1,003	520	581	351
V4	8,493	7,501	6,520	6,500	12	0	168	63	49	39	1,010	454	733	445
V5	6,414	5,304	4,200	4,200	120	0	109	107	103	77	1,090	484	792	436
V6	19,672	13,987	12,712	11,200	4,180	1,300	730	323	84	136	1,185	607	781	421

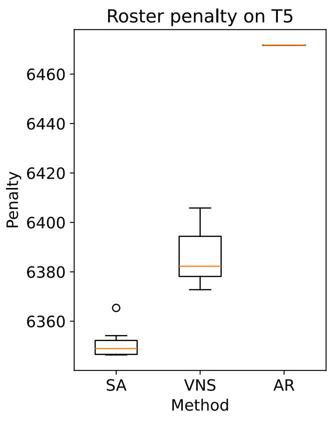
Table D.14: Run time and roster properties for Construction-per-shift (CPS) and Simulated Annealing (SA) on all nine instances, averaged over 25 runs for CPS, over 10 runs for SA. The numbers are rounded somewhat for clarity. Roster property abbreviations are shown in Table D.8.

	RT		**US**		**UPS**		**OHPE**		**UR**		**PSSLR**		**PDOSLR**	
Instance	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**	**CPS**	**SA**
O4	0.54	600.16	181	177	0.0	0.0	3.83	3.38	27.4	16.0	0.13	0.53	0.32	0.71
O5	0.57	600.15	229	228	0.0	0.0	2.98	2.69	24.3	20.0	0.12	0.54	0.32	0.71
O6	0.52	600.15	205	203	0.4	0.0	3.88	3.38	20.0	17.0	0.12	0.52	0.34	0.74
T4	0.10	600.06	28	24	3.0	3.0	4.11	4.49	5.3	0.6	0.16	0.54	0.45	0.64
T5	0.12	600.05	29	26	7.0	7.0	2.48	1.99	0.6	0.0	0.13	0.56	0.48	0.70
T6	0.11	600.05	40	36	11.0	11.0	4.26	4.67	3.6	0.0	0.11	0.55	0.49	0.71
V4	0.09	600.06	13	13	13.0	13.0	0.56	0.17	7.3	6.0	0.11	0.63	0.37	0.66
V5	0.08	600.05	10	10	8.0	8.0	0.36	0.36	12.4	10.0	0.11	0.58	0.37	0.67
V6	0.14	600.06	55	40	18.0	18.0	2.36	1.50	6.4	10.2	0.13	0.52	0.43	0.69

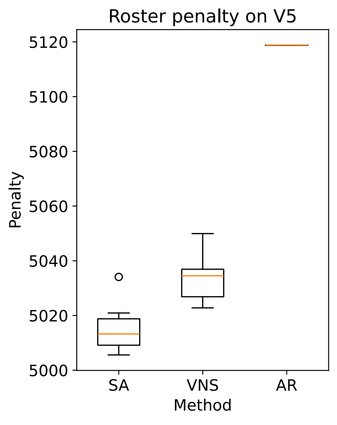
D.3. AutoRoster comparison results



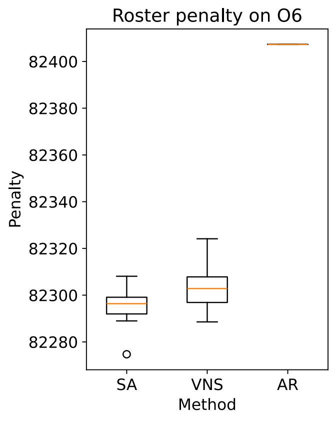
(a)



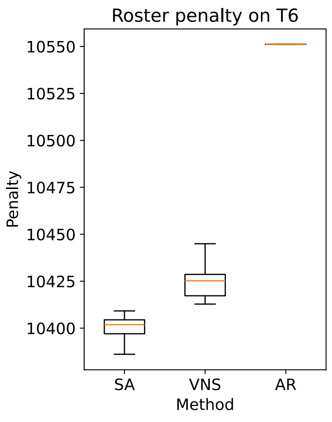
(b)



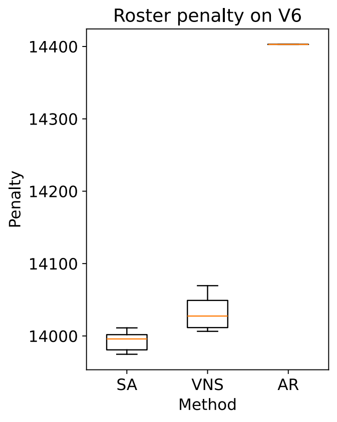
(c)



(d)



(e)



(f)

Figure D.10: Box plots of the results of Simulated Annealing (SA) and Variable Neighborhood Search (VNS) compared to the AutoRoster (AR) local search algorithm on the modified model, shown in the AutoRoster column of Table 6.4, on the May and June instances over 10 runs per instance.

Table D.15: Comparison of the total penalty (TP), coverage penalty (CP) and coverage spread penalty (CSP) results of Simulated Annealing (SA), Variable Neighborhood Search (VNS) and the AutoRoster local search algorithm (AR) on the modified model, shown in the AutoRoster column of Table 6.4, on all nine instances, averaged over 10 runs per instance. The numbers are rounded to integers for clarity.

	TP			**CP**			**CSP**		
Instance	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**
O4	64,813	64,814	64,999	17,700	17,700	17,700	43,500	43,500	43,500
O5	98,602	98,605	98,844	22,800	22,800	22,800	72,800	72,800	72,800
O6	82,295	82,303	82,407	20,300	20,300	20,300	58,800	58,800	58,800
T4	5,598	5,614	5,734	3,600	3,600	3,600	0	0	0
T5	6,351	6,386	6,472	5,400	5,400	5,400	0	0	0
T6	10,400	10,425	10,551	8,000	8,000	8,000	600	600	600
V4	7,252	7,269	7,355	6,500	6,500	6,500	0	0	0
V5	5,015	5,034	5,119	4,200	4,200	4,200	0	0	0
V6	13,993	14,032	14,403	11,200	11,200	11,400	1,300	1,300	1,400

Table D.16: Comparison of the overtime hours spread penalty (OHSP), requests penalty (RP), preferred shift sequence length penalty (PSSLP) and preferred day-off sequence length penalty (PDOSLP) results of Simulated Annealing (SA), Variable Neighborhood Search (VNS) and the AutoRoster local search algorithm (AR) on the modified model, shown in the AutoRoster column of Table 6.4, on all nine instances, averaged over 10 runs per instance. The numbers are rounded somewhat for clarity. Abbreviations of the penalty categories are shown in Table D.7.

	OHSP			**RP**			**PSSLP**			**PDOSLP**		
Instance	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**
O4	2,205.41	2,206.93	2,210.32	171.0	171.0	179.0	732	734	864	505	502	545
O5	1,493.89	1,495.73	1,511.60	220.0	220.0	228.0	784	787	968	504	502	537
O6	2,077.41	2,079.24	2,075.72	178.0	178.8	178.0	575	591	687	365	354	367
T4	1,205.20	1,205.20	1,205.28	4.4	6.6	37.0	437	454	524	352	348	368
T5	319.02	319.97	319.02	0.0	0.0	0.0	385	421	488	247	245	265
T6	1,185.63	1,185.63	1,185.63	0.0	0.0	0.0	387	394	496	228	246	269
V4	62.94	63.02	63.06	39.0	39.0	44.0	305	327	371	344	340	377
V5	106.74	107.33	105.95	77.0	77.0	80.0	340	357	439	292	292	294
V6	327.59	355.09	293.60	128.8	143.6	159.0	614	607	727	422	427	423

Table D.17: Roster properties related to the different soft constraint penalties after application of Simulated Annealing (SA), Variable Neighborhood Search (VNS) and the AutoRoster local search algorithm (AR). The values are averaged over 10 runs and rounded somewhat for clarity. The abbreviations for the different roster properties are clarified in Table D.8.

	US			**UPS**			**OHPE**			**UR**			**PSSLR**			**PDOSLR**		
Instance	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**	**SA**	**VNS**	**AR**
O4	177	177	177	0	0	0	3.37	3.38	3.40	16.0	16.0	17.0	0.53	0.52	0.42	0.73	0.73	0.70
O5	228	228	228	0	0	0	2.70	2.70	2.75	20.0	20.0	21.0	0.54	0.54	0.44	0.72	0.72	0.69
O6	203	203	203	0	0	0	3.38	3.39	3.37	17.0	17.1	17.0	0.53	0.52	0.44	0.73	0.74	0.73
T4	24	24	24	3	3	3	4.50	4.50	4.50	0.4	0.6	3.0	0.61	0.60	0.54	0.71	0.72	0.70
T5	26	26	26	7	7	7	1.99	2.00	1.99	0.0	0.0	0.0	0.64	0.60	0.54	0.75	0.76	0.72
T6	36	36	36	11	11	11	4.67	4.67	4.67	0.0	0.0	0.0	0.63	0.62	0.52	0.77	0.76	0.74
V4	13	13	13	13	13	13	0.17	0.17	0.18	6.0	6.0	7.0	0.67	0.66	0.60	0.69	0.70	0.66
V5	10	10	10	8	8	8	0.36	0.37	0.34	10.0	10.0	11.0	0.63	0.61	0.54	0.71	0.71	0.70
V6	40	40	42	18	18	18	1.50	1.56	1.33	9.7	10.7	12.0	0.52	0.53	0.44	0.69	0.68	0.69