TU WIEN Informatics

A Learning Large Neighborhood Search for the Staff Rerostering Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Fabio Francisco Oberweger, BSc
Matrikelnummer 01551139

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Docent Elina Rönnberg, PhD
Projektass. Marc Huber, MSc

Wien, 13. Oktober 2021

Fabio Francisco Oberweger	Günther Raidl

TU WIEN Informatics

A Learning Large Neighborhood Search for the Staff Rerostering Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Fabio Francisco Oberweger, BSc
Registration Number 01551139

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Docent Elina Rönnberg, PhD
Projektass. Marc Huber, MSc

Vienna, 13th October, 2021

Fabio Francisco Oberweger	Günther Raidl

Erklärung zur Verfassung der Arbeit

Fabio Francisco Oberweger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13. Oktober 2021

Fabio Francisco Oberweger

Danksagung

Zuallererst möchte ich mich bei meiner Familie, allen voran Mama Birgit, Papa Ralf, Angelo, Karin, Jasmin, Reinelde und Camilla, bedanken.

Auch bedanken möchte ich mich bei meinen Betreuer:innen Günther Raidl, Elina Rönnberg und Marc Huber für ihre Unterstützung und Feedbacks über die vergangenen Monate.

Diese Arbeit war Teil der Vorstudie *Decision support for railway crew planning*, die von KAJT (Kapacitet i Järnvägstrafiken/Kapazität im Eisenbahnverkehr), einem schwedischen Forschungsprogramm zur Verbesserung der Leistung von Eisenbahnsystemen, unterstützt wird.

Acknowledgements

First and foremost, I would like to thank my family, especially Birgit, Ralf, Angelo, Karin, Jasmin, Reinelde, and Camilla.

I also want to thank my supervisors Günther Raidl, Elina Rönnberg, and Marc Huber for their support and feedback over the past months.

This work has been part of the pre-study *Decision support for railway crew planning* supported by KAJT (Kapacitet i Järnvägstrafiken/Capacity in the Railway Traffic System), a Swedish research programme for improved railway system performance.

Kurzfassung

In dieser Arbeit stellen wir eine mit *Machine Learning* (ML) erweiterte *Large Neighborhood Search* (LNS) vor, um das *Staff Rerostering Problem* (SRRP) zu lösen. Das SRRP ist ein kombinatorisches Zeitplanungsproblem, das sich mit Störungen eines bestehenden Arbeitsplans befasst, z. B. Krankenstand von Arbeitnehmer:innen oder Änderung des Personalbedarfs. Das Ziel des SRRPs ist es, einen neuen Arbeitsplan unter Berücksichtigung dieser Störungen zu erstellen und so wenige Änderungen wie möglich am ursprünglichen Plan vorzunehmen.

Eine LNS besteht aus sich wiederholenden Anwendungen einer Zerstör- und einer Reparaturmethode. Zuerst hebt die Zerstörmethode die zugewiesenen Werte einer Teilmenge von Entscheidungsvariablen (Zerstörmenge) auf und fixiert die Werte der anderen Variablen. Danach versucht die Reperaturmethode, die vorherige Lösung durch das Suchen von besseren Werten für die Variablen aus der Zerstörmenge zu verbessern. Der Hauptbeitrag dieser Arbeit ist eine ML-basierte Zerstörmethode. Wir trainieren ein *Conditional Generative Model*, das eine Wahrscheinlichkeitsverteilung lernt. Diese Verteilung gibt an, welche Variablen in die Zerstörmenge aufgenommen werden sollten. Um hochwertige Zerstörmengen aus diesen Wahrscheinlichkeiten zu erstellen, präsentieren wir eine auf das SRRP zugeschnittene Strategie. Das verwendete Neuronale Netz ist ein *Graph Neural Network* (GNN), das als Eingabe einen Graphen verwendet, der eine SRRP-Instanz modelliert. Wir wenden *Imitation Learning* an, um das Neuronale Netz zu trainieren und dabei ein spezielles gemischt-ganzzahliges lineares Programm (MILP) nachzuahmen. Dieses MILP berechnet optimale Zerstörmengen. Es benötigt aber zu viel Zeit, um in tatsächlichen Anwendungen verwendet zu werden. Die Reparaturmethode unserer LNS besteht darin, ein vorgeschlagenes MILP für das SRRP zu lösen.

Unsere ML-basierte LNS liefert mehr als doppelt so gute Resultate in Bezug auf die durchschnittliche Optimalitätslücke als das Lösen des SRRP-MILPs mit Gurobi. Sie übertrifft sogar die Ergebnisse einer leistungsstarken LNS, ausgestattet mit einer manuell gestalteten Zerstörmethode. Sehr bedeutungsvoll ist, dass die ML-basierte LNS auf unterschiedliche Zeitpläne mit unterschiedlicher Anzahl von Mitarbeitern generalisieren kann und auch die anderen Ansätze hierbei deutlich übertrifft.

Abstract

We propose a large neighborhood search (LNS) enhanced with machine learning (ML) for the staff rerostering problem (SRRP). The SRRP is a combinatorial timetabling problem and deals with disruptions to an existing schedule, e.g., illness of employees or change in demand for staff. The goal of the SRRP is to construct a new schedule considering these disruptions and introducing as few changes as possible to the original schedule.

An LNS consists of repeated applications of a destroy and a repair operator. First, the destroy operator induces a subproblem by unassigning a subset of decision variables (destroy set) and fixing the others. Then, by solving this subproblem, the repair operator tries to improve the previous solution by finding better assignments for the unassigned variables. The main contribution of this thesis is an ML-based destroy operator. We train a conditional generative model to estimate probabilities indicating which variables should be destroyed. We propose a refined sampling strategy tailored to the SRRP to build high-quality destroy sets from these probabilities. Our model is a graph neural network (GNN), which takes a custom graph modeling an SRRP instance as an input. To train our neural network, we apply imitation learning to mimic a mixed-integer linear program (MILP) based approach that can compute optimal destroy sets but is far too expensive to use in actual applications. We utilize an additional GNN to learn optimal parameters for the destroy set sampling process. The repair operator of our LNS consists of solving a proposed MILP.

Our learning-based LNS outperforms solving the MILP with Gurobi by a factor greater than 2.65 in terms of optimality gap. It even surpasses the results of a well-performing LNS with a meaningful manually designed destroy operator in all respects. Most importantly, it generalizes to different schedules with various numbers of employees and also comfortably outperforms the other approaches on this test set.

Contents

CHAPTER 1

Introduction

This thesis deals with solving the staff rerostering problem (SRRP), a combinatorial timetabling problem, with a machine learning (ML) enhanced large neighborhood search (LNS). The motivation of this work is presented in Section 1.1. In Section 1.2, we discuss the aim of this thesis. Lastly, we outline the following chapters in Section 1.3.

1.1 Motivation

Our motivation regarding this thesis is two-fold. First, we seek to contribute to the research concerning the integration of ML techniques into traditional state-of-the-art combinatorial optimization methods. We believe that learning-based strategies have the potential to improve the performance of optimization algorithms, considering that hardware components and neural network architectures have become increasingly powerful over the last few years. Concerning hard combinatorial optimization problems (COP), end-to-end learning approaches are, in general, still clearly outperformed by conventional state-of-the-art optimization methods. Therefore, we propose a hybrid approach embedding an ML model into an LNS meta-heuristics for improved guidance of the heuristic search. Second, we aim to efficiently solve the SRRP, for which we show that the ML-enhanced LNS is a highly suitable choice. Efficiently solving scheduling problems has a considerable impact on social aspects such as the well-being of employees but also heavily influences the economic capacities of institutions.

For most companies and institutions, it is crucial to meet the given demand as best as possible at any moment. In the industrial sector, for example, supplies need to be available on time. In public transport, transport links must be operated so that employees can commute to work. In healthcare, an adequate number of caregivers needs to be provided to the patients. If employees are absent, shortages can occur that prevent the demand from being met. Reasons for these absences might be overwork, illness, injury, childcare obligations, or other appointments. If the minimum required staff is

not available, a work schedule requires rerostering. For the healthcare domain, Moz and Pato [MP03] defined the nurse rerostering problem (NRRP), which occurs when one or more nurses cannot cover their previously assigned shifts. In this thesis, we adapt the NRRP to a more general version, which we call the SRRP.

The SRRP is an extension of the classical nurse rostering problem (NRP). Similar to the NRP, it deals with constructing work schedules for employees. Employees shall be assigned to shifts or may have off-days. Only specific shift sequences are allowed to ensure the work roster complies with labor laws. For example, there may be the requirement that there must be an eleven-hour rest between two shifts and that the total number of workdays has to be within a specified range. There are constraints on how many consecutive shift assignments in total and per shift type are allowed. In this work, we specifically consider early, day, and night shifts. The extension of the SRRP to the NRP lies in the fact that the SRRP has to deal with employee absences since employees cannot work shifts if they are absent. This has an impact on the objective of the problem as well. The goal is to create a roster that covers the demands per shift and day, assigns the employees an approximately even workload, and incorporates employee preferences as well as possible. While this aim could be the same for the NRP, for the SRRP, we have the additional goal to make the new schedule as similar as possible to the original one. In contrast to the NRRP, we also consider demand disruptions, which means that the staffing requirements for specific shifts may change in the SRRP. For a more extensive overview of the SRRP, see Section 2, and for a formal definition of the problem, we refer to Section 5.

Moz and Pato [MP07] proved that already the NRRP with the single objective of minimizing the differences from the original schedule is NP-hard. Traditionally, it is distinguished between three main approaches to solve NP-hard optimization problems: exact algorithms, approximation algorithms, and heuristics. Since exact algorithms, which are typically based on branch-and-bound, may not scale well to large instances and approximation algorithms provide quality guarantees that often are too weak for practical purposes or have scalability issues in respect to practical runtimes, often heuristics are used in practice. In this thesis, our goal is to find high-quality solutions fast since rerostering needs to be done at short notice, and relatively quick decisions are vital. Hence, we propose an LNS [PR10] meta-heuristics, which Pisinger et al. [PR10] argue to be highly successful in routing and scheduling applications, to solve the SRRP. Since SRRP instances often are composed similarly in one real application, we enhance the LNS by an ML-based destroy operator trained to exploit this structure.

1.2 Methodological Approach

Typically the LNS applies a destroy and repair operator combination in each iteration. First, it partially destructs the incumbent solution by freeing a subset of the decision variables while fixing the others to their current values. A subproblem is hereby induced and its space of feasible solutions forms a (large) neighborhood. By exactly or heuristically

solving this subproblem, the LNS tries to improve the previous solution by finding better assignments for these "destroyed" variables and fixing the others. If a new solution with a better objective value than the incumbent is found, it is the starting point for the next destroy and repair iteration. The LNS repeats this procedure until a stopping criterion is reached.

If the sub-problems created by the destroy operator are small enough, they can be solved efficiently by effective exact approaches. Our LNS approach relies on a mixed-integer linear program (MILP) which we solve with the Gurobi[1] solver as a repair method. The more challenging aspect is to design the destroy operator and therefore an appropriate neighborhood structure. Standard approaches for destroy operators are problem-specific fast heuristics and randomized methods. However, designing and choosing effective methods often is time-consuming and still gives no guarantee for a successful solution method. Recently, learning-based techniques have been applied to COPs for discovering patterns that are hard to find by hand. These strategies reduce or even eliminate the need to manually construct heuristics and have the potential to unveil connections that humans might not see.

Thus, this thesis aims to design an ML-based LNS consisting of a learning-based destroy operator and a MILP as a repair method. We apply imitation learning to train a conditional generative model predicting probabilities indicating which variables should be destroyed. Based on these probabilities, we propose a sampling strategy tailored to the SRRP for an appropriate selection of these variables. For this sampling strategy, we introduce an additional neural network producing suitable parameters steering the sampling process. As our neural network architectures, we employ graph neural networks (GNN), for which we provide a custom graph structure representing an SRRP instance. A GNN [GMS05, SGT+08] is a neural network architecture that is independent of the graph size and therefore also problem instance size and able to represent the underlying structure of a problem. Compared to existing methods, our custom graph structure enables efficient learning and inference, facilitating the application of ML to highly constrained COPs such as the SRRP.

In addition to our custom learning LNS, we design a conventional LNS that relies on a meaningful manually designed destroy method and the same MILP-based repair operator. We compare our learning-based LNS approach to this standard LNS and solving the defined MILP directly with Gurobi. To the best of our knowledge, no SRRP benchmark instance dataset that meets our requirements is available. Hence, we created a custom collection of benchmark instances to test our methods. The learning-based LNS, trained on a schedule with 110 employees and multiple different sets of disruptions, outperforms solving the MILP with Gurobi by a factor greater than 2.65 on average in terms of optimality gap. It also improves the optimality gaps achieved by the handcrafted LNS by more than 1% on average. Most importantly, the learning-based LNS shows the same performance on different original schedules with various (unseen) numbers of employees and surpasses the results of the handcrafted LNS in all respects.

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

1.3 Outline of the Thesis

In Section 2, we provide an extensive overview of the SRRP. Afterward, in Section 3, we review the literature regarding staff rostering problems and ML in combinatorial optimization, including existing work about neural LNSs with a learned neighborhood destruction method [ANA20, SLYD20, CGCL20, SWK+21]. In Section 5, we formalize the SRRP and model it as a MILP. Section 6 contains the used LNS framework consisting of the employed construction heuristic, repair operator, and manually designed destroy method. The main section of this thesis is Section 7, where we introduce our learning-based destroy operator and all associated details, such as the training data generation. In Section 8, we provide details on the test instance generation and present the results obtained in computational results. Finally, Section 9 concludes this thesis by summarizing the achievements and giving an outlook on promising future work.

CHAPTER 2

The Staff Rerostering Problem

The SRRP is a combinatorial timetabling problem and closely related to the NRP. The main difference is that the SRRP deals with disruptions to an existing work schedule instead of creating a new one. In the literature, the SRRP is usually called NRRP. This name, however, is connected to the healthcare sector and might implicitly indicate some domain-specific properties. For example, the number of nurses needed for a shift usually does not change. At least, this has not been considered in NRRPs so far [MV11, MV13, MP03, MP04, MP07, WSB19]. In other industry sectors, changes in demand are more frequent, e.g., due to vehicle problems or track work in the railway domain. Moreover, the number of nurses employed in a hospital unit frequently is smaller than the number of employees in other industry branches. Thus, we aim to keep the problem general and refer to it as SRRP.

We first specify the problem statement of the NRP since the SRRP builds on top of this definition. Moreover, this approach allows us to emphasize the differences and similarities of these problems in more detail. The NRP deals with constructing a periodic work schedule for a set of heterogeneous employees. More specifically, each employee shall be assigned to one possible shift from a set of shifts or an off-day on each day in this period. This final work schedule should then meet diverse constraints. Burke et al. [BDCBVL04] identified the parameters and constraints most commonly used in the NRP literature and summarized them in the following way:

- *Planning period.* The planning period determines the time frame over which the employees are scheduled.

- *Skill category.* Specifies the qualification or skill set of employees. Particular shifts may require different qualifications or skills.

- *Shift type.* Shift types usually determine the start and end times of a shift. Most frequently, NRPs deal with the traditional three 8-hour shifts. These are the early shift (e.g., 7 am – 3 pm), day shift (3 pm – 11 pm), and night shift (11 pm – 7 am).

- *Coverage constraints.* These state the demand for employees and specific skill-sets for every single shift in the planning horizon.

- *Time-related constraints.* Time-related constraints include the availability of employees, shift preferences of employees, and constraints on demanding an equal workload among the staff.

- *Work regulations.* They stem from the contracts of the employees, labor laws, or other institutional regulations. Possible constraints falling in this category are the minimum rest between working shifts, the maximum number of working shifts in the planning period, and similar ones.

All these constraints are usually defined as hard or soft constraints. A solution is infeasible if the hard constraints are not satisfied. On the contrary, it is possible to violate soft constraints but these violations are penalized in the objective function. For example, an employee might request a day off. To improve employee satisfaction it is encouraged to grant this request. But if it is not reasonable or even impossible to consider this request in the schedule, violating it is still allowed. However, disregarding soft constraints leads to penalization in the optimization objective, which reduces the quality of the solution as a consequence. Thus, the interaction of the soft constraints plays a vital role in the solution-finding process. To account for the different importances of soft constraints, weights are assigned to them.

As we have previously mentioned, the SRRP builds on top of the problem statement of the NRP. Hence, the already stated properties and constraints hold for this problem as well. Additionally, the SRRP deals with the occurrence of disruptions to an already created roster. For this reason, the classical rostering problem has to be solved in advance of the SRRP to obtain this initial schedule. A disruption occurs if an employee is unable to perform assigned shifts or if the number of required employees in a shift has changed on one or more days. We distinguish between three different types of disruptions:

- *Single-shift disruption.* This type of disruption occurs when an employee is unable to cover a single shift, e.g., because of a medical or otherwise necessary appointment.

- *Multi-shift disruption.* They arise if an employee is unable to perform more than one consecutive working shifts. Reasons for multi-shift disruptions can, for example, be illness or vacation.

- *Demand disruption.* These disruptions occur when the required number of employees unexpectedly changed on one or more days.

The goal of the SRRP is to modify the original roster to incorporate these changes. In addition to the constraints arising from the classical rostering problem, the disruptions lead to two new constraints. The first constraint is time-related. It specifies that employees cannot be assigned to a shift if they are absent at the time of this shift. The second constraint deals with the changes made to the original roster. These should be as small as possible for the convenience of the employees. Again, this new schedule should, similarly to the NRP, fulfill all the hard constraints and meet the soft constraints as well as possible.

For our specific SRRP definition, we build on a modified version of the NRRP by Maenhout and Vanhoucke [MV11], and also we rely on their instance generation algorithm [VM09]. The details of the test instance generation are covered in Section 8.1. We consider a three-shift setting, where the early shift lasts from 7 am to 3 pm, the day shift from 3 pm to 11 pm, and the night shift from 11 pm to 7 am. Additionally, we add a free shift indicating that an employee does not have to work that day to model off days. We refer to the free shift as a non-working shift and the other as working shifts. In the following, we state the hard (prefix H) and soft constraints (prefix S) that we consider for the SRRP. We provide a formal description of the problem in Section 5.

HOSPD (one shift per day) Each employee can only be assigned to one working shift per day or the free shift.

HREST (minimum rest) An employee has to rest at least 11 hours after each working shift.

HWSA (working shift assignments) An employee must not be assigned to less than a minimum or more than a maximum number of working shifts in the scheduling period.

HCWSA (consecutive working shift assignments) An employee must not be assigned to less than a minimum or more than a maximum number of consecutive working shifts.

HSTA (shift type assignments) An employee must not have less than a minimum or more than a maximum number of assignments to a shift type in the scheduling period.

HCSTA (consecutive shift type assignments) An employee must not have less than a minimum or more than a maximum number of consecutive assignments to a shift type.

HABSE (employee absences) Employees cannot be assigned to a working shift if they are absent for the time of this shift on this day.

SCREQ (cover requirements) The staffing requirements per day and shift should be met.

SMOD (modifications to orignal roster) The original roster should be modified as little as possible.

SPREF (employee preferences) Employee preferences should be taken into account.

SEWL (even workload) The workload should be distributed as evenly as possible among the employees.

Hard constraints HOSPD, HREST, HWSA, HCWSA, HSTA, and HCSTA are time-related constraints. These enforce that the new roster is compliant with labor laws and other institutional regulations. Moreover, the constraints having a minimum and maximum can be used to ensure the social quality of the schedule. For example, an employee assigned to a long sequence of night shifts might find it hard to maintain social contact. Hard constraint HABSE is a rerostering-specific constraint. An employee can be unable to do a shift for several reasons such as vacation, illness, or medical appointments. Absences due to vacation or illness might even spread over multiple days. The first soft constraint SCREQ is a so-called coverage constraint. Staffing requirements are often classified as a soft constraint since it is possible to engage external replacement or use less personnel in a shift. Both of these variants might come at an extra cost. On the one hand, using fewer employees than required could decrease customer satisfaction or slow down production. On the other, hiring short-term staff at short notice is more expensive. Another goal of the rerostering problem is to modify the original roster as little as possible (SMOD) since employees build their life around the duty schedule. Hence, too many changes might decrease employee satisfaction. Not only modifications to the original roster but also neglected shift preferences can lower employee contentment, which is the focus of soft constraint SPREF. Finally, constraint SEWL deals with fairness. It should ensure that employees should more or less have an even workload over the planning period.

CHAPTER 3

Related Work

In this section, we discuss the related work relevant to this thesis. First, we give a literature overview of classical rostering problems such as the NRP. Second, scientific work concerning rerostering problems is reviewed. And finally, we discuss different strategies, how ML has been applied in the literature to solve COPs, including methods dealing with the introduction of ML into algorithmic templates such as the LNS.

3.1 Staff Rostering Problems

Staff rostering, or scheduling, is a problem that has been addressed by computer scientists and operations researchers for decades. It deals with the construction of work schedules for the employees of an organization such that different objectives are achieved. These objectives might be to reduce costs, fulfill demands, increase employee well-being, enhance job satisfaction, or a combination thereof [EJKS04]. Many companies from different fields, such as the healthcare, transportation, or industrial sector, are affected by these goals. Since there are various requirements or unique needs in each domain, many staff rostering problem variants exist. Ernst et al. [EJKS04] give an overview of rostering problems in different application areas and the solution methods proposed in the literature.

Since the efficient utilization of time and effort is crucial in the healthcare domain, the NRP, also called nurse scheduling problem, is strongly represented in research [BDCBVL04, CLLR03]. Burke et al. [BDCBVL04] and Cheang et al. [CLLR03] give an overview of models and solution methods in their surveys dedicated to the NRP. Similar to the general staff rostering problem, many different variants of the NRP arose over time. Therefore, Van den Bergh et al. [VdBBDB+13] categorize papers based on the characteristics of the considered problems. They perform these categorizations within different fields such as personnel attributes, constraints, solution method, and application area. For example, in the field of personnel attributes, they classify problems based on whether skills are considered or not, amongst other criteria.

This variety of NRPs makes it hard to compare various solution approaches. Vanhoucke and Maenhout [VM09] observed this lack of unified formulations and benchmark instances. Thus, they proposed an algorithm to generate NRP instances based on selected complexity indicators. These indicators contain the number of employees, the length of the scheduling period, and the number of possible shifts. Additionally, they include values determining the distributions of the coverage requirements and employee preferences for working specific shifts. More benchmark problems and instances were introduced in the First [HDCSS14] and Second International Nurse Rostering Competition [CDDC$^+$19].

Driven by the practical relevance of the NRP, researchers have proposed several solution methods for this problem. Especially MILP approaches have shown popularity in research. In addition to exactly solving MILPs with branch-and-price procedures [LOR19, MV10], hybrid MILP-based strategies have often been applied successfully [LOR19, BLQ10, GTS17, STGR16, BC14, VGG$^+$12]. Frequently, these hybrids consist of combining integer programming with local search techniques. For example, Burke et al. [BLQ10] use a MILP to solve a sub-problem containing all hard and only a subset of soft constraints. Then, they improve this solution by applying a variable neighborhood search (VNS). Santos et al. [STGR16] use a greedy heuristic to generate an initial solution and then enhance it by utilizing a MILP-based VNS. Particularly successful is the hybrid approach of Valouxis et al. [VGG$^+$12], which won the First International Nurse Rostering Competition [HDCSS14]. They apply MILPs in two stages of their algorithm to achieve certain sub-goals. These consist of determining the workload of nurses and distributing the daily shifts. Additionally, they improve the solutions with different local searches.

In this work, we apply a LNS [Sha98]. The LNS is a powerful meta-heuristics for solving COPs and has been successfully applied to various complex problems, including the vehicle routing problem (VRP) [Sha98], facility location problem (FLP) [JORR20], and the resource-constrained project scheduling problem (RCPSP) [Mul09]. Furthermore, Pisinger and Ropke [PR10] claim that the LNS heuristics has been most successful in routing and scheduling applications. Syed et al. [SAKB19] even name LNS as the state-of-the-art method for VRP. The LNS usually consists of a destroy and repair operator combination that is applied to a solution repeatedly. An extension of the LNS is the adaptive large neighborhood search (ALNS) which consists of multiple destroy and repair operators. In the ALNS, the operators are newly selected in each iteration based on their previous performances.

LNSs have also been applied in the context of nurse scheduling in the literature. Bilgin et al. [BDCRB12], for example, applied an ALNS with a tabu list to solve the NRP. They represent the solution as a set of (nurse, day, shift type, skill type)-tuples to indicate an allocation of a nurse to a specific shift. In contrast to the classical LNS approach with operators that destroy or repair variable assignments of a solution, the strategy of Bilgin et al. [BDCRB12] is slightly different. They apply various methods similar to local searches that either remove, add, or modify single tuples of a solution. In the case a tuple is only changed, they argue that this operator is a destroy and repair operator in one method. Another ALNS approach for solving the NRP was proposed by Legrain et

al. [LOR19]. They apply three randomized destroy operators that work in the classical sense. The first destroy method randomly selects nurses from the set of all nurses, the second one nurses having the same skill, and the last one nurses with the same contracts. For each operator, they destroy the schedules of the selected nurses for a specific number of weeks. Similarly to the chosen destroy operator, this number of weeks is determined based on its effectiveness in previous iterations. The destructed solutions are repaired by a MILP that is solved with a branch-and-price procedure that Legrain et al. [LOR19] also present in their paper.

Often expected or unexpected events lead to the need that an existing schedule has to be adapted. Most of the time, these events are employee-related and can be due to illness, vacation, or appointments. Moz and Pato [MP03] were the first researchers that formally defined the NRRP. The NRRP deals with creating a new employee schedule given absences of the staff on specific days. Therefore, it is an extension of the NRP focusing on the reconstruction of disrupted work schedules. Moz and Pato [MP07] proved that the NRRP is NP-hard given the single objective of minimizing the differences from the original schedule. The NRRP is not as widespread in research as the NRP but has been considered in several works [MV11, MV13, MP03, MP04, MP07, WSB19, PM08].

Moz and Pato [MP03] formulated a multi-commodity flow model with the single objective of reducing the deviations from the original work roster. Furthermore, they solved the integer linear programming formulation of this model and created a construction heuristic to reconstruct the schedule instantly. However, this construction heuristic does not guarantee feasible solutions. Later, Moz and Pato [MP04] formulated a new multi-commodity flow model with decreased size and showed that the corresponding integer program could outperform the existing version. In addition to integer programming, Moz and Pato [MP07] applied a genetic algorithm to solve the NRRP. In this paper, they also improved the construction heuristic from [MP03]. To test their algorithms, Moz and Pato [MP03, MP04, MP07, PM08] relied on real-world instances from a hospital in Lisbon. Pato and Moz [PM08] also were the first to address the NRRP with multiple objectives. The two goals were to create a schedule with an even workload between nurses and to incorporate preferences. Therefore, they designed a bi-objective genetic heuristic.

Similar to Moz and Pato [MP07], Maenhout and Vanhoucke [MV11] also proposed a genetic algorithm to solve the NRRP. In contrast to the previously discussed papers, however, they considered multiple objectives such as minimizing under- and overstaffing, promoting an even workload, and including employee preferences as well as possible. They claim that their approach outperforms the genetic algorithm of Moz and Pato [MP07]. To compare the algorithms, they modified Moz and Pato's algorithm to make it compatible with their constraints and objectives. Furthermore, Maenhout and Vanhoucke [MV11] investigated whether it is necessary to contemplate the whole schedule during the reconstruction. They found out that to get reasonable solutions fast, it suffices to consider parts of the roster only. They confirmed this in a follow-up study [MV13] where they claim that the days before and after the disruptions are the most important during re-optimization. Additionally, they argue that it is sufficient to consider the affected

nurses and a subset of other nurses when rerostering.

Finally, Wickert et al. [WSB19] proposed multiple solution approaches. Like Maenhout and Vanhoucke [MV11], they consider an NRRP with multiple objectives. First, they formulated a MILP that is, in contrast to the flow model from Moz and Pato [MP03, MP04], based on an assignment problem. Second, they designed a variable neighborhood descent meta-heuristics to avoid the dependency from a commercial solver. To obtain test instances, they introduced random disruptions to the NRP instances of the Second International Nurse Rostering Competition [CDDC+19]. Additionally, they used the Lisbon hospital instances from Moz and Pato [MP03].

3.2 Machine Learning in Combinatorial Optimization

In recent years many researchers have been investigating the application of ML to COPs. The goal is to automate (part of) the demanding process of handcrafting algorithms for these types of problems. Since a successful approach to automate this process would lead to significant savings in time or other resources, several different techniques and methods have been proposed.

One strategy found in research is centered around the application of supervised machine learning [VFJ15, LCK18]. Vinyals et al. [VFJ15], e.g., proposed a sequence-to-sequence model called Pointer-Network. The Pointer-Network is an adaptation of the long short-term memory model (LSTM) [HS97] by Hochreiter et al., enabling the model to work on inputs of variable lengths. To achieve this, the network points back to an input value, which is selected based on the outputs of a softmax function. Vinyals et al. [VFJ15] applied supervised machine learning to train their Pointer-Network and solved small instances of the traveling salesperson problem (TSP) amongst other problems.

To successfully apply supervised machine learning to COPs, however, optimal or high-quality solutions are needed in the training phase. For NP-hard problems those optimal solutions are difficult to find. Therefore, supervised learning approaches are limited to small instances or have to rely on a non-optimal ground truth, which hurts the solution quality. To overcome this issue, reinforcement learning (RL) [SB18] is a promising technique [BPL+16, NOST18, MDM+20, KDZ+17, AXSS19, HPD19]. In a reinforcement learning environment an agent automatically explores solutions and learns from received feedback. Thus, no ground truth is required in this setting.

Bello et al. [BPL+16] built on top of the work of Vinyals et al. [VFJ15] and combined Pointer-Networks with a reinforcement learning training regime to solve the 2D Euclidean TSP. They apply the asynchronous advantage actor-critic (A3C) [MBM+16] algorithm to optimize the parameters of their model. The actor-critic approach aims to stabilize the reinforcement learning training process by introducing an extra model, the critic, which evaluates the actions of the main model, the actor. The parameters of the actor are then updated based on the evaluations of the critic. Once the model is trained, Bello et al. [BPL+16] sample multiple solutions based on the softmax outputs of the model

and took the best. With this approach, they could outperform the results of Vinyals et al. [VFJ15] and report near-optimal solutions for TSP instances with up to 100 nodes.

Despite these improvements, Pointer-Networks still suffer from limitations. One limitation is that Pointer-Networks consider sequences as inputs and outputs. Therefore, they automatically force an ordering on the solution. Many problems such as the TSP, however, are independent of any order. Cappart et al. [CCK+21] refer to this limitation as "invariance to permutation" in their work. Furthermore, it is hard to effectively reflect the underlying structure of a problem with Pointer-Network models. A neural network architecture family that addresses and solves these issues is the GNN [GMS05, SGT+08]. Graph neural networks have been around for over a decade but have gained high popularity in recent years due to the need for structured representations and computations in learning, as described by Battaglia et al. [BHB+18]. The main idea behind these networks is to create a node's representation by aggregating its own and neighbors' features [WPC+20]. Graph neural networks can be categorized into four groups after Wu et al. [WPC+20]: recurrent GNN (RecGNN) , convolutional GNN (ConvGNN), spatial–temporal GNN (STGNN), and graph autoencoder (GAE). RecGNNs and ConvGNNs are the most frequently used architectures in the context of combinatorial optimization. Wu et al. [WPC+20] define the main difference between these two groups in their distinct use of the learnable parameters. Whereas RecGNNs use the same parameters over all their layers, ConvGNNs have a different set of parameters in each layer.

The combination of graph neural network and reinforcement learning has been applied by several researchers in some form [KDZ+17, AXSS19, ANA20] and currently seems to be one of the most promising strategies for solving combinatorial optimization problems with machine learning methods. The first researchers to employ this approach in the context of combinatorial optimization problem were Khalil et al. [KDZ+17].

Khalil et al. [KDZ+17] argue that most COPs can be transformed into problems on graphs and therefore focus on solving combinatorial graph problems. They applied a structure2vec (S2V) [DDS16] GNN architecture belonging to the RecGNN class and paired it with Q-learning [WD92]. Q-learning is a RL method where actions are not selected directly but indirectly through estimated average future rewards. Their approach, called S2V-deep Q-network (DQN), imitates a greedy heuristics. It generates solutions by iteratively applying the trained model to the problem and adding the node with the highest score to the solution until it is feasible. Khalil et al. [KDZ+17] showed that the S2V-DQN outperforms basic heuristics and approximation algorithms on graphs with up to 500 nodes for the minimum vertex cover problem (MVC), maximum cut problem (MAXCUT), and TSP. Moreover, they reported that they outperformed the Pointer-Network approach of Bello et al. [BPL+16] in their experiments. To obtain similar inference times, Khalil et al. [KDZ+17] let the Pointer-Network greedily choose the nodes based on their softmax scores instead of the sampling method mentioned above. Finally, they showed that the S2V-DQN trained on instances with 50 and 100 nodes could still reach an approximation ratio less than 1.11 on test instances with up to 1200 nodes for all mentioned problems.

Reinforcement learning research has also been driven forward by the interest and investigations of many researchers in the domain of combinatorial games. AlphaGo Zero [SSS+17] by Silver et al., for example, is a superhuman engine of the ancient Chinese board game Go. This learning engine attracted attention in this research community as Go has a large search space. Moreover, many people believed that it required human intuition to be successful in this game. Later, Silver et al. generalized the AlphaGo Zero algorithm to AlphaZero [SHS+18] and showed that it could also learn other combinatorial games such as Chess and Shogi more efficiently than different approaches. The AlphaGo Zero and AlphaZero RL algorithms combine self-play with Monte Carlo Tree Search (MCTS), a combination of tree search and random sampling [BPW+12], to train a deep neural network.

Due to its success in combinatorial games, Alpha Zero also aroused the interest of researchers in the field of combinatorial optimization [AXSS19, HPD19]. Huang et al. [HPD19] combined AlphaZero with a special neural network architecture and high-performance computing to solve the graph coloring problem. They claim to have learned new state-of-the-art heuristics. Abe et al. [AXSS19], on the other hand, considered a more general strategy. Similar to Khalil et al. and their S2V-DQN they focused on graph formulations for COPs as well. Therefore, Abe et al. applied the AlphaZero RL algorithm to train GNNs. Instead of selecting a single GNN model, they tested various architectures. These were S2Vs [DDS16], graph convolutional networks (GCN) [KW17], graph isomorphism networks (GIN) [XHLJ19], and 2-IGN+ [MBHSL19]. Except for the S2Vs, all networks fall into the class of ConvGNNs. In their experiments, they considered the MVC, MAXCUT, and the maximum clique problem (MAXCLIQUE). Due to its long inference times, the 2-IGN+ frequently did not finish within the time limit. But for the other architectures, Abe et al. [AXSS19] showed that their approach outperforms the S2V-DQN. Moreover, when using MCTS during test-time, they could even surpass the best-known solutions for seven instances of MAXCLIQUE. The application of MCTS during test-time increases inference time significantly, however.

Machine learning techniques that learn heuristics for COPs in an end-to-end fashion have been steadily improving over recent years. Nonetheless, they are usually still outperformed by state-of-the-art classical optimization methods. This especially holds for problems with complex side constraints. Consequently, to close this performance gap a new paradigm often referred to as "learning to search" [SLYD20] has evolved in research. This paradigm deals with the introduction of automatically learned heuristics in algorithmic templates. Therefore, eliminating the need for manually created algorithms. For example, learnable heuristics were used for guiding beam search [NGG18], deciding on how to branch in branch-and-bound (B&B) algorithms [GCF+19, HDIE14, KBS+16], and learning destroy or repair operators in LNS meta-heuristics [ANA20, SLYD20, CGCL20, SAKB19, HT19].

Especially the research around learning how to branch in exact algorithms such as B&B seemed to be promising. However, this approach failed to deliver practical impact. Song et al. [SLYD20] claim that the main reason for this lies in the requirement to modify commercial solvers like CPLEX and Gurobi, which only offer limited and inefficient

interfaces. Moreover, they mention the possibility of altering open-source solvers such as SCIP [Ach09]. These solvers, however, are significantly slower than leading commercial solvers making this strategy unsuitable as well [SLYD20]. Therefore, researchers have shifted their attention to powerful meta-heuristics such as the LNS, which consists of an alternating application of destroy and repair operators. So far, these operators were manually created heuristics, which often were randomized or built after greedy criteria. But as previously mentioned, the rise of ML in combinatorial optimization has driven forward investigations on how to use learning to reduce or even eliminate the need for handcrafted heuristics.

One way of integrating machine learning into the LNS framework is to learn the repair operator. Syed et al. [SAKB19] and Hottung et al. [HT19] chose this strategy in their research. Both approaches rely on manually created destroy methods and focus on training a repair method to solve the resulting sub-problems. Similarly, both works apply slightly modified variants of the Pointer-Network. Syed et al. [SAKB19] employed an actor-critic reinforcement learning strategy, whereas Hottung et al. [HT19] used supervised learning. Finally, both showed that their approaches outperform the classical LNS with handcrafted repair heuristics.

In the design of the LNS, sub-problems often are made small enough such that highly optimized general purpose MILP solvers such as CPLEX or Gurobi can be used as repair methods [JORR20]. Song et al. [SLYD20] and Addanki et al. [ANA20] followed this approach. They learned destroy-heuristics that automatically select a subset of variables that are then unassigned. Subsequently, they rely on the MILP solver for repairing the solutions.

Song et al. [SLYD20] proposed a general LNS framework for solving MILPs. They used a decomposition-based LNS. In this version of the LNS, all variables are split into disjoint sets in each iteration. Then, each variable set is destroyed and repaired one after the other, concluding one iteration of the LNS. Song et al. [SLYD20] trained a model intending to learn these decompositions for a given solution. They experimented with three different learning approaches: behavior cloning [Pom88] and forward training [RB10], which both belong to the class of imitation learning, and classical RL, applying the basic REINFORCE [Wil92] algorithm. The behavior cloning method trains the model on samples created by experts in a supervised manner. Contrarily, forward training is closer to RL. It collects samples on its own by applying the experts' actions in each state. Hence, there exists a relation to the predicted decomposition of previous states. Instead of consulting an expert, Song et al. [SLYD20] randomly sampled multiple decompositions for each solution and took the best. They could show that their approach outperforms a LNS with random decompositions in the destroy method. Furthermore, Song et al. [SLYD20] report that their LNS variants find equally good solutions between two to ten times faster than Gurobi for the MVC and MAXCUT amongst other problems. Regarding the different learning algorithms, basic reinforcement learning performed worst. The reason for this, however, could be the out-dated REINFORCE algorithm.

Another instance of "learning to search" is the work of Addanki et al. [ANA20]. Similar

to Song et al. [SLYD20], they suggested a general framework for solving MILPs as well. They called their approach neural LNS and also rely on a MILP solver as the repair operator. The framework of Addanki et al. [ANA20], however, has two key differences from the framework of Song et. al. [SLYD20]. First, Song et. al. [SLYD20] did not specify which ML model they were using. But based on their explanations it seemed to be a model that can only handle inputs of the same dimensions. In comparison, Addanki et al. [ANA20] applied a message-passing neural network (MPNN) which can be classified as a ConvGNN after Wu et al. [WPC+20]. They obtained a graph structure called Constraint-Variable Incidence Graph (CVIG) from the MILP constraints by introducing a node for each variable and constraint, then adding an edge between them if the variable occurs in the constraint. The weight of the edge is equal to the factor of the variable in the constraint. Second, instead of having a model that predicts decompositions, the MPNN of Addanki et al. [ANA20] only infers single nodes. They apply the MPNN to the solution, unassign the selected variable, and repeat this process until they "destroyed" the desired amount of variables. As their learning method of choice, Addanki et al. [ANA20] chose the V-trace algorithm [ESM+18], which is yet another actor-critic strategy. Like other works in the learning LNS niche, they showed that their approach could outperform LNSs with randomized and other MILP-based destroy methods.

In recent works, Nair et al. [NBG+20] and Sonnerat et al. [SWK+21] abandoned RL and focused on supervised ML as a conditional generative modeling task. Although Nair et al. [NBG+20] did not consider an LNS in their research, their proposed Neural Diving algorithm still turned out to be a promising starting point for the work of Sonnerat et al. [SWK+21] and this thesis. In principle, Neural Diving is a primal heuristic represented by a neural network model. This model is trained to approximate a conditional distribution that, given a MILP, generates probabilities that can be used to sample high-quality decision variable assignments. As training data, Nair et al. [NBG+20] collected all the feasible solutions that a MILP solver found during solving the training instances. They designed their learning task such that better solutions are assigned a higher probability and therefore have a higher weight in the loss function, and worse solutions have a lower probability and weight. Eventually, they applied their model to sample multiple partial assignments, employed the MILP solver to find values for the unassigned variables, and reported the best solution found as a result. An important aspect of their work is that, in contrast to Addanki et al. [ANA20], they perform their predictions for all variables simultaneously instead of one neural network evaluation per variable. This approach significantly speeds up computations at test time. However, due to the required assumption of conditional independence, some accuracy is lost when approximating the conditional distribution. Nonetheless, this is a reasonable trade-off in the domain of heuristic optimization.

As we have already mentioned, Sonnerat et al. [SWK+21] built on top of Nair et al. [NBG+20] and proposed a learning-based LNS. Their applied generative modeling approach conditions on a current solution in an LNS run and defines a distribution over destroy sets. Unlike Nair et al. [NBG+20], they only assign a probability of one to the

best destroy set for a solution in the defined conditional distribution. To create training data, they used an expert policy to generate LNS trajectories. This expert policy consists of their main MILP in combination with local branching [FL03], which only allows the solution to change in at most η variables. By then comparing the old solution with the newly created one, they can extract optimal destroy sets. During test time, Sonnerat et al. [SWK+21] first apply their model to generate probabilities for each variable. Then, they iteratively select variables to be destroy proportionally to their probabilities until the maximum capacity of the destroy set is reached. Similar to Nair et al. [NBG+20], they also employed a CVIG to model their problems and a GCN-based neural network architecture. Both, Nair et al. [NBG+20] and Sonnerat et al. [SWK+21] reported strong empirical results outperforming or tying classical approaches on multiple MILP instance datasets.

CHAPTER 4

Methodology

In this section, we give an overview of the theoretical foundations and optimization methods applied in our algorithm. We begin by defining what an optimization problem is in Section 4.1. Then, we focus on heuristics in Section 4.2, where we also discuss the LNS. In Section 4.3, we explain the main concepts of mathematical programming. Afterward, we deal with the ML domain in Section 4.4, where we focus on RL and GNNs especially.

4.1 Optimization Problems

We follow Papadimitriou and Steiglitz [PS98]. Researchers usually distinguish between two categories of optimization problems. These are continuous and discrete optimization problems. As their names indicate, they either deal with problems involving continuous or discrete variables. Discrete optimization problems, also called COPs, are typically concerned with finding a set, permutation, or graph structure from a finite or countably infinite input set. The problem we consider in this thesis, the SRRP, is a COP since its decision variables are discrete, and the goal is to find a set of shift assignments for each employee. Optimization problems stand in contrast to satisfaction problems. Satisfaction problems deal with the question of whether there is a feasible solution at all, e.g., "Is it possible to deliver all customers in one route?". An optimization problem, on the contrary, aims to find not just one but the best solution w.r.t. a given objective (cost) function. Hence, the respective question could be "What is the shortest route to deliver all customers?". To formalize these intuitions, we state the definition of an optimization problem by Papadimitriou and Steiglitz [PS98] in Definition 1.

Definition 1 *An optimization problem is a set of problem instances. An instance is a pair* $(\mathcal{S}, c)$*, where* $\mathcal{S}$ *is the set of all feasible solutions (the solution space), and* c *is a cost function* $c : \mathcal{S} \to \mathbb{R}$*. The problem is to find a solution* $s \in \mathcal{S}$*, such that for all* $y \in \mathcal{S}$ *it*

holds that $c(s) \leq c(y)$. Such a solution s is called a globally optimal or simply an optimal solution.

Note that in this definition, the problem is viewed as a minimization problem since s is an optimal solution if its cost value $c(s)$, also called objective value, is smaller or equal to the value $c(y)$ of all other solutions $y \in \mathcal{S}$. Nevertheless, Definition 1 also holds for maximization problems. The reason for this is that one can transform any maximization problem into a minimization problem by simply multiplying its cost function by minus one.

A naive approach for solving COPs is to enumerate all feasible solutions $s \in \mathcal{S}$, compare their objective value $c(s)$, and take the best solution. This procedure falls under the category of exact methods as it guarantees the optimality of the found solution. While this naive approach works for small and simple problems, it usually does not work well for hard problems, such as the SRRP. For those, the solution space is in general too large, making it intractable to iterate over all the feasible solutions.

A more sophisticated exact method is B&B, which relies on the idea of intelligently enumerating feasible solutions. Let $(\mathcal{S}, c)$ be an instance of a discrete minimization problem and s^* its optimal solution. Then, a value $d \in \mathbb{R}$ is called dual bound (lower bound) if and only if $d \leq c(s^*)$. Oppositely, a value $p \in \mathbb{R}$ is called primal bound (upper bound) if and only if $c(s^*) \leq p$. Thus, the objective value $c(f)$ of a feasible solution $f \in \mathcal{S}$ is always a primal bound. Once a solution $s \in \mathcal{S}$ is found such that $p = d = c(s)$ holds, then s is guaranteed to be an optimal solution. The B&B method stores the best solution found and its objective value, which automatically is the best upper bound found, during the whole procedure. The algorithm works by dividing the solution space into multiple mutually exclusive subproblems (branching). For example, for a problem with a set of binary variables X, some unsassigned variable $x \in X$ is set to 0 for one subproblem and to 1 for the other. Then, upper and lower bounds are computed for these subproblems (bounding). If the lower bound is higher than the upper bound of the global best solution, the algorithm discards this subproblem since no better solution can be found by expanding it. Otherwise, the procedure later branches on the subproblem again. The algorithm stops once the lower bounds for all subproblems are higher than or equal to the globally best upper bound, or all feasible solutions have been visited.

Hence, in the worst-case, the B&B algorithm also requires a complete enumeration. In practice, however, it often achieves substantial accelerations compared to more naive approaches. Note that for a maximization problem, the dual bound is an upper bound, and the primal bound is a lower bound. Furthermore, the B&B procedure works the same only the lower and upper bounds are used the other way around.

4.2 Heuristics

Another class of approaches for solving COPs is the application of heuristic and meta-heuristic methods. For the review on these, we use Blum and Raidl [BR16] as a basis. In

contrast to exact methods, heuristics give no guarantee for a specific solution quality. As we have discussed previously, exact approaches ensure the optimality of the generated solution. This guarantee comes at the cost of extensive computation times too high for practical applications, especially for hard and real-world problems. Heuristics offer an alternative way of generating high-quality, even near-optimal solutions in an acceptable amount of time in exchange for this guarantee of optimality. Hence, they are generally applied for solving COPs in practice since reasonably good solutions are often sufficient. In this review, we focus on construction heuristics, the local search, and the LNS. We do not directly apply a local search in our solution algorithm. However, the basic concepts of the local search, such as the definition of a neighborhood, are essential for the LNS as well.

4.2.1 Construction Heuristics

Construction heuristics are used to create initial solutions and also serve as a foundation for other heuristic algorithms, which further operate on this solution. They generally work by iteratively adding components to an initially empty solution until it is complete or one cannot extend it further. Hence, they have the name construction heuristics as they construct solutions step by step from scratch. Moreover, they are typically fast but usually leave room for improvement. One of the most common strategies for construction heuristics is the greedy strategy. Here, a greedy function evaluates the potential components to be added to the solution from a local perspective. Then, the component with the highest value is greedily selected. Again, this procedure is repeated until a solution is complete or not extendable.

4.2.2 Local Search

In contrast to construction heuristics, local search does not generate solutions from scratch. Instead, it operates on an already existing solution, which depicts its starting point. Then, the local search iteratively replaces the current solution with a better solution found in an appropriately defined neighborhood of the current solution. It repeats this procedure until the neighborhood of the current solution does not contain a solution which is better than the current one. In Definition 2, we state Blum and Raidl's [BR16] definition of a neighborhood, which more formally is a neighborhood function. Remember that an instance of an optimization problem is defined as $(\mathcal{S}, c)$, where $\mathcal{S}$ is the solution space and $c : \mathcal{S} \to \mathbb{R}$ the cost function.

Definition 2 *A neighborhood function $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$ assigns to each solution $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \in \mathcal{S}$, which is called the neighborhood of s.*

The goal of the local search is to reach a local minimum, which is a solution $\hat{s} \in \mathcal{S}$ such that $\forall s \in \mathcal{N}(\hat{s}) : c(\hat{s}) \leq c(s)$ holds with respect to a neighborhood function $\mathcal{N}$. A local minimum is not necessarily a globally optimal solution as previously defined. Otherwise, the local search would be an exact method as well. Another essential aspect of the local

search is the selection of the new solution in the neighborhood. This new solution must be better than the current solution, but there can be more than one such solution. Most commonly, two so-called step functions are applied to handle this selection. These are the first improvement and the best improvement step functions. Whereas the first solution which is better than the current solution is selected in the first improvement strategy, the best improvement method searches the whole neighborhood and takes the best solution. In Algorithm 1, we show the pseudocode for a basic local search procedure.

Algorithm 1: Local Search

Input: initial solution s.
while $\exists s' \in N(s) : c(s') < c(s)$ *and termination criterion not met* **do**
  $s \leftarrow$ `StepFunction`$(N(s))$
end
return s

4.2.3 Large Neighborhood Search

The LNS proposed by Shaw [Sha98] belongs to the class of meta-heuristics [PR10]. Meta-heuristics are high-level and problem-independent algorithmic frameworks or templates for creating heuristic optimization algorithms. They aim at combining construction heuristics, local search methods, or both with other concepts. One essential property of meta-heuristics is that they all have integrated mechanisms for escaping local optima. Therefore, they can explore the solution space more flexibly, enhancing the solution quality of such approaches. The LNS achieves this by increasing the size of the neighborhood and using more suitable algorithms to find better solutions instead of iterating through all the neighbors. Prominent meta-heuristics are ant colony optimization, variable neighborhood search, simulated annealing, tabu search, and genetic algorithms, to name a few.

In this thesis, we design an algorithm based on the LNS meta-heuristics, which we review in this section. In addition to Blum and Raidl [BR16], we orient ourselves on Pisinger and Ropke [PR10] for this review. As we have already mentioned in previous sections, the LNS consists of a destroy and a repair method, also called destroy and repair operators. While the destroy method $d(\cdot)$ partially destructs a current solution, the repair operator $r(\cdot)$ rebuilds the destructed solution aiming to improve it. The neighborhood $\mathcal{N}(s)$ for a solution $s \in \mathcal{S}$, cf. Definition 2, is therefore implicitly represented by the destroy and repair methods since $N(s)$ is a set of solutions that can be reached by applying $r(d(s))$.

Algorithm 2 shows the principle of LNS in pseudocode. First, an initial solution is created by some construction heuristic for a problem instance $(\mathcal{S}, c)$, where $\mathcal{S}$ is the solution space and $c : \mathcal{S} \to \mathbb{R}$ the cost function. Then, in each iteration, the incumbent solution s^{current} is partially destroyed by removing some of its components resulting in s^{partial}. The number of components to remove, called the degree of destruction, is an important parameter of the LNS. Besides, the way how to choose these components is an important

Algorithm 2: Large Neighborhood Search

Input: problem instance $I = (\mathcal{S}, c)$.

$s^{\text{current}} \leftarrow$ `GenerateInitialSolution`(I)
$s^{\text{best}} \leftarrow s^{\text{current}}$
while *termination criterion not met* **do**
  $s^{\text{partial}} \leftarrow d(s^{\text{current}})$
  $s^{\text{new}} \leftarrow r(s^{\text{partial}})$
  if $c(s^{new}) < c(s^{best})$ **then**
    $s^{\text{best}} \leftarrow s^{\text{new}}$
  end
  $s^{\text{current}} \rightarrow$ `ApplyAcceptanceCriterion`$(s^{\text{new}}, s^{\text{current}})$
end
return s^{best}

design choice as well. Often, destroy methods include some stochasticity such that the probability of destructing different components in each iteration is higher. After every application of the destroy method, the LNS invokes the repair operator to create a new solution s^{new} from s^{partial}. A powerful technique to achieve good solutions is to rely on highly optimized solvers, such as MILP solvers. Since only parts of the problem are destroyed, and most of the solution remains fixed, the complexity of the subproblems is considerably smaller, and exact solvers may be effective again. Stopping the solver before it reaches optimality is often a valid strategy because the optimized components might change again in the next iteration. The LNS automatically stores the best solution s^{best} reached so far. For the acceptance of a new incumbent solution, an acceptance criterion can be applied which chooses between s^{new} and s^{current}. Possibilities are to select the better solution or use a metropolis criterion as applied in simulated annealing.

4.3 Mathematical Programming

We base this overview on mathematical programming on Blum and Raidl [BR16], Schrijver [Sch98], and Wolsey et al. [WN99]. Mathematical programming has many facets. In this section, we focus on linear modeling approaches such as integer linear programs (ILP), MILPs, and linear programs (LP). Here, the task is to formulate and solve mathematical models for problems having a linear objective function and a set of linear constraints. Especially ILPs and MILPs are popular approaches for solving COPs in practice. The reason for this is a combination of the ease of creating a valid model and the existence of powerful generic solvers such as CPLEX, Gurobi, or SCIP.

An ILP can in general be stated in the following form

$$z_{\text{ILP}} = \min\{c^\top x \mid Ax \geq b,\, x \geq 0,\, x \in \mathbb{Z}^n\} \tag{4.1}$$

where x is an n-dimensional integer variable vector of domain $\mathbb{Z}^n$ and $c \in \mathbb{Q}^n$ is an

n-dimensional vector of constants, often referred to as costs. The objective function of the model is defined by the dot product $c^\top x$ of the costs and variables. The goal of an ILP is to find a variable assignment for x that minimizes the objective value without violating the given constraints. In (4.1), these constraints are represented by the inequalities based on coefficient matrix $A \in \mathbb{Q}^{m \times n}$ and vector $b \in \mathbb{Q}^m$. In this ILP definition, we considered a minimization problem. To translate a maximization into a minimization problem and vice versa, solely the sign of c has to be changed. Similarly, by changing the sign of the corresponding coefficients, one can transform greater-than-or-equal-to inequalities into less-than inequalities. Equality constraints can be converted into pairs of inequalities. Hence, without loss of generality, we only consider here models of the stated form.

In contrast to ILPs, MILPs also allow continuous variables in addition to discrete variables. We state a formal definition of a MILP in (4.2), where y is an additional non-negative p-dimensional continuous variable vector, $d \in \mathbb{Q}^p$ the cost vector for variables y, and $B \in \mathbb{Q}^{m \times p}$ the coefficient matrix for the constraints. The other variables and constants are defined equally as for (4.1).

$$z_{\text{MILP}} = \min\{c^\top x + d^\top y \mid Ax + By \geq b,\, x \geq 0,\, y \geq 0,\, x \in \mathbb{Z}^n\} \tag{4.2}$$

Solving ILPs and MILPs is NP-hard [Pap81]. General-purpose solvers such as CPLEX and Gurobi belong to the class of exact methods. Therefore, they apply highly optimized methods based on B&B, as sketched in Section 4.1, to solve integer programs. Relaxations are a fundamental concept in this context. A problem is relaxed if some constraints are loosened or omitted. The relaxed problem is typically easier to solve than the original one. The solutions of relaxed problems are not necessarily feasible for the original problem but represent bounds of it. An LP relaxation of an integer program is obtained by relaxing its integrality constraints. Thus, the domain of the integer variable vector is changed from $\mathbb{Z}^n$ to $\mathbb{R}^n$. In (4.3), we show the LP relaxation of the MILP in (4.2). The solution of the relaxed program is a lower bound of the respective MILP, which means that $z_{\text{LP}} \leq z_{\text{MILP}}$. Hence, B&B methods for solving MILPs are often based on LP relaxations used to compute the lower bounds in a B&B node.

$$z_{\text{LP}} = \min\{c^\top x + d^\top y \mid Ax + By \geq b,\, x \geq 0,\, y \geq 0,\, x \in \mathbb{R}^n\} \tag{4.3}$$

The reason why LP relaxations can in general be solved more efficiently than integer programs is that LPs are in P. Therefore, it is possible to find optimal solutions in polynomial time. Examples of polynomial-time algorithms are the ellipsoid method [Kha80] and the interior point method [Kar84]. However, the ellipsoid algorithm provides poor performance in practice. Even though this is not the case for interior point approaches, the simplex method by Dantzig [Dan51] is still the most frequently used algorithm in practice. Despite its exponential worst-case complexity, the simplex algorithm is usually superior to other methods.

4.4 Machine Learning

For this review of ML, we follow Murphy [Mur12], Goodfellow et al. [GBCB16] and Géron [Gér19], unless stated otherwise. Since there are several manifestations of ML, we start this section by giving a generic definition by Mitchell [M$^+$97], determining what learning in the context of computer programs means:

Definition 3 *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*

Usually, researchers divide ML into three sub-fields. One of these sub-fields is supervised learning. In supervised ML, the goal is to learn a mapping from inputs $x \in X$ to outputs $y \in Y$, given a labeled input dataset $D = \{(X_i, Y_i)\}_{i=1}^{N}$. Here, D, also called the training set, corresponds to the experience E from Definition 3 and contains N training samples or instances. The structure of the training inputs X may vary from task to task. On the one hand, they can be p-dimensional vectors of numbers, called features or attributes, in the simplest case. On the other, they might represent complex objects such as images or graphs. There are two types of supervised learning depending on the domain of the labels $y \in Y$. If y is categorical or nominal, meaning that $y \in \{1, \ldots, C\}$ for some number of classes C, the problem is known as classification. On the contrary, if y is real-valued, it is called a regression. The mentioned mapping is expressed as a function f_θ, which depends on a set of parameters θ. In the context of ML, this function f_θ is called a model, and its parameters are called weights. There also are non-parametric models which we do not consider in this review. During the learning process, the model repeatedly receives input instances $(x, y) \in D$ and computes the similarity between its output $\hat{y} = f_\theta(x)$ and the actual label y with a so-called error or loss function. Popular loss functions for the regression and classification are the mean squared error and the cross-entropy loss, respectively. Then, the weights θ of the model are updated based on this score (loss) to improve the future performance of the model. Methods to update the weights are usually gradient descent-based approaches such as stochastic gradient descent (SGD) or momentum optimizers. Gradient descent methods operate by computing the gradients of the loss function for the weights θ and updating the weights in the direction of the descending gradients. Finally, similar to Definition 3, a performance measure T is consulted to evaluate the progress and the final performance of the model. If a model is too expressive and extensively trained on the

The other two sub-fields of ML are unsupervised learning and RL. In supervised learning, there exists a ground truth represented by the labels Y. In unsupervised learning and reinforcement learning, however, there are no labels. Hence, the given inputs are of the form $D = \{X_i\}_{i=1}^{N}$ in an unsupervised learning setting. Here, the goal is to find patterns in the data without specifying these patterns or defining a specific error function. Examples of unsupervised learning tasks are principal component analysis and clustering.

4.4.1 Reinforcement Learning

In this overview of RL, we follow Sutton and Barto [SB18] and Zai and Brown [ZB20]. As previously mentioned, there is no labeled data in the context of RL and hence no data that can be directly used to train a model. Therefore, the setting of RL consists of an agent and an environment. The agent takes actions in the environment and moves from one state to another. Based on these actions and state transitions, the agent receives positive or negative rewards, which it uses to learn from and make better decisions in the future. Customarily, RL problems formally are modeled as a Markov decision process (MDP). MDPs were introduced by Howard et al. [How60] and are represented by a 4-tuple (ST, A_s, T, R) where ST is a set of possible states, A_s is a set of available actions from state s, $T : ST \times A_s \to ST$ is a state transition function, and $R : ST \times A_s \to \mathbb{R}$ is a reward function. Figure 4.1 shows such a described agent-environment interaction in an MDP.

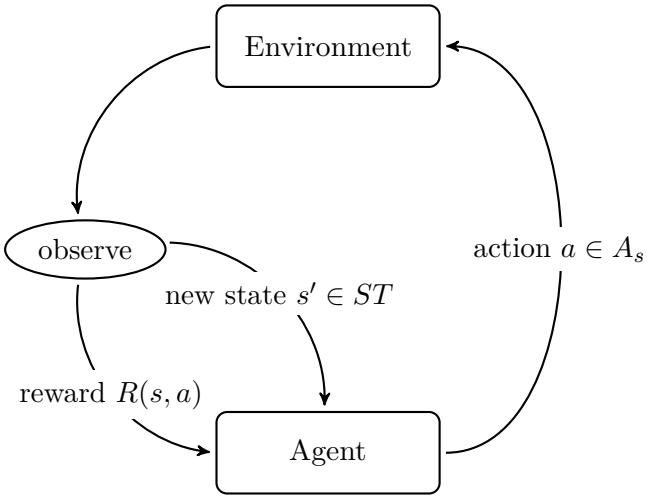


Figure 4.1: The agent–environment interaction in a Markov decision process. Adapted from [SB18, p. 48].

Q-Learning MDPs are the formal foundation for modeling RL problems, but they do not define how the agent learns from its actions in the environment. For this reason, Watkins et al. [WD92] introduced the Q-learning approach, "which provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions". Standard Q-learning is based on a table that holds so-called Q-values $Q(s, a)$ for each state-action pair $(s, a) \in ST \times A_s$. A Q-value $Q(s, a)$ represents the average reward upon leaving state s with action a, plus expected later rewards. When the agent then takes action a in state s of the environment at time step t, the Q-values are updated by applying the following formula:

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot$$

$$\overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)}^{\text{learned value}} . \tag{4.4}$$

The learning rate $\alpha \in (0, 1]$ in (4.4) regulates how quickly the new Q-value adjusts to the learned value. Hence, too small values for α lead to slower convergence, and too large values might lead to divergence. The parameter $\gamma \in [0, 1]$ determines the importance of the future values. Therefore, γ is an important ingredient to handle the temporal difference (TD) of the rewards. Formula (4.4) shows the update based on 1-step TD learning since it is based on just one next reward. An extension to this approach would be n-step TD learning. The idea is to wait for n steps before updating the Q-values to get a more accurate estimate of the future rewards. Using Q-learning and given enough time, the Q-values will eventually converge to their optimal values if the agent takes a random action in each state [WD92]. Usually, an epsilon-greedy strategy, a combination of random actions and greedily choosing the best action based on Q-values, is used during training to improve exploration of the state-action space.

Due to memory issues, however, tabular Q-learning methods cannot deal with problems that consist of a vast amount of state-action pairs. Hence, Mnih et al. [MKS+13] were the first to apply deep-Q-networks, which take a state s and action $a \in A_s$ as input and approximate the Q-values for such state-action pairs. These deep-Q-networks are trained based on the difference between the old and new Q-value and rely on deep learning and neural networks, which we both cover in Section 4.4.2.

Policy Gradient Methods In Q-learning, actions are indirectly selected based on their expected rewards, the Q-value. However, there also exist methods where a network directly outputs a probability distribution over the actions. Such methods are called policy gradient methods. Here, a policy function π_θ with parameters θ takes a state $s \in ST$ as an input and for all actions $a \in A_s$, returns a probability $\pi_\theta(a|s)$ indicating the preference of this action in this state. In modern applications, neural networks approximate these policy functions. One of the most famous policy gradient learning method is the REINFORCE algorithm by Williams [Wil92]. Since this algorithm serves as a basis for many other methods in this area, we state its pseudocode in Algorithm 3.

The REINFORCE algorithm is also called Monte Carlo Policy Gradient. In RL, the term "Monte Carlo" not only refers to methods with an essential random component but also to methods performing updates only after the termination of an episode. Episodes are repeated interactions in an agent-environment setting, e.g., plays of a game. Moreover, a trajectory is a sequence of state-action-reward triples (s_t, a_t, r_{t+1}) collected by an agent in each time step $t \in \{1, \ldots, T-1\}$ during the execution of an episode. These triples represent the rewards r_{t+1} that the agent received for taking action a_t in state s_t at time step t. An episode begins independently of how its previous episode ended, and eventually, they all terminate in a state called terminal state. Since Monte Carlo methods wait until the completion of an episode before updating the parameters, they represent

Algorithm 3: REINFORCE: Monte Carlo Policy Gradient

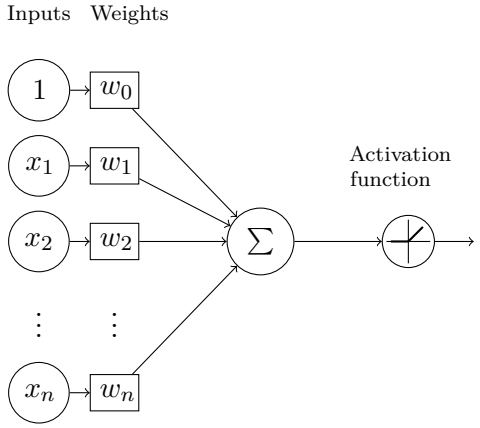
Input: a differentiable policy function $\pi_\theta(a|s)$, learning rate α, discount rate γ.

$\theta \leftarrow$ `InitializeParameters()`
while *termination criterion not met* **do**
  $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T \leftarrow$ `GenerateEpisode`(π_θ)
  foreach *step of the episode* $t = 1, \ldots, T-1$ **do**
    $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    $\theta \leftarrow \theta + \alpha \gamma^t G_t \Delta \ln \pi_\theta(a_t|s_t)$
  end
end
return θ

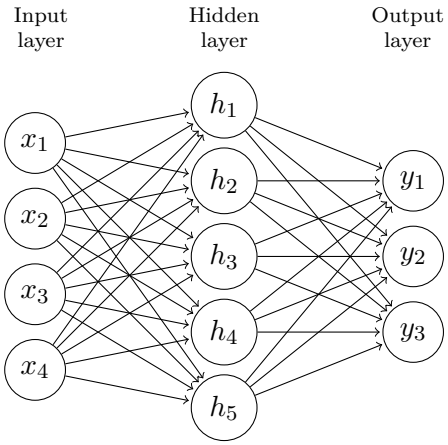
the counterpart of 1-step TD learning. The n-step TD strategy lies between those two approaches. Returning to the REINFORCE procedure shown in Algorithm 3, the first step deals with initializing the parameters. Then, the algorithm enters the training loop, where a trajectory is generated with respect to the current policy $\pi_\theta(\cdot|\cdot)$, meaning that an agent executes an episode based on its current state of "knowledge". Next, REINFORCE iterates over each time step $t \in \{1, \ldots, T-1\}$. First, it computes G_t representing the total sum of discounted future rewards starting at time step t. Subsequently, it updates the weights θ based on the gradients of the logarithmized model output $\pi_\theta(a_t|s_t)$ and the discounted G_t value considering the learning rate α and the discount rate γ. This update method is based on the Policy Gradient Theorem, for which we refer to Sutton and Barton [SB18, p. 325]. The intuition behind this update strategy is that the parameters are pulled in such a direction that the model assigns higher probabilities to actions that received higher rewards in a specific state and vice versa.

4.4.2 Neural Networks & Deep Learning

An important concept in the artificial neural network domain is the perceptron introduced by Rosenblatt [Ros57]. Perceptrons aim to model biological neurons that receive signals as inputs, sum this information up with different weights, and fire if this sum exceeds a certain threshold. Mathematically, the perceptron is defined as a function $p_\theta(x) = \sigma(x^\top \theta + b)$, where $x \in \mathbb{R}^n$ is the n-dimensional input, $\theta \in \mathbb{R}^n$ the vector of weights, b a bias constant, and $\sigma : \mathbb{R} \to \mathbb{R}$ a non-linear activation function. Popular activation functions are the sigmoid, rectified linear unit (ReLU), and hyperbolic tangent (Tanh) functions. In Figure 4.2a, we show a visual representation of a perceptron. The expressiveness of perceptrons is limited, e.g., they are unable to model exclusive-or functions. Therefore, they are seldomly used as a single model for practical applications. In the same way as the brain, artificial neural networks do not consist of a single neuron either. Instead, they get their power from the interaction of a collection of them. Thus, the multi-layer perceptron (MLP) [Ros57], a neural network model, uses perceptrons as its most essential component. We show an example of an MLP with four inputs, one hidden layer with



(a) Visualization of a perceptron [Ros57]. Each input x_i, including a bias constant, is multiplied with a corresponding weight w_i. The summation node sums up the resulting products, and the final value is "activated" by a non-linear activation function.



(b) Multi-layer perceptron (MLP) with four-dimensional inputs, one hidden layer with five nodes and three outputs. The arcs represent the weights. Bias, summation, and activation nodes are omitted for clarity in this MLP.

Figure 4.2: Visualizations of a perceptron and a MLP.

five nodes, and three outputs in Figure 4.2b. In an MLP, each hidden and output layer node is the output of an embedded perceptron. Since each such perceptron comes with its own set of parameters, researchers struggled to find an appropriate way to train the MLP successfully for several years. Then, Rumelhart et al. [RHW85] introduced the backpropagation training algorithm in 1985. This algorithm combines gradient descent with an efficient method for automatically computing the gradients for each parameter and is still used to train neural networks today. The standard gradient descent algorithm is SGD, but more sophisticated algorithms such as ADAM [KB17] exist. The MLP is also called a feed-forward network, as the information flows from the input to the output layer without entering any feedback loops.

Together with the MLP, convolutional neural networks (CNN) and recurrent neural networks (RNN) are the most frequently used classical neural network architectures. We introduced the MLP because it is seen as the basis for more advanced models having significant implications for practical application. The CNN, which is most successful in object and image recognition, is a specialized version of the MLP. Furthermore, the RNN, which empowers many language processing or speech recognition applications, builds upon its concepts. Also, more complex models frequently use MLPs as an additional component.

The main difference between deep learning and classical ML is that a deep learning problem is expressed as a composition of multiple, possibly even different functions.

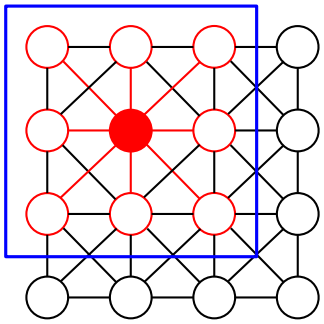
Hence, a model in the deep learning context is a function of the following form

$$f(x) = f_{\theta_k}^{(k)}(\dots(f_{\theta_2}^{(2)}(f_{\theta_1}^{(1)}(x))) \tag{4.5}$$

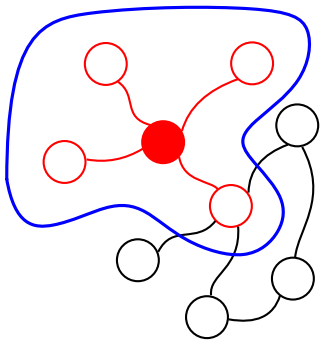
where $k \geq 2$ and for each depth level $l \in \{1, \dots, k\}$ of the model, there is a function $f_{\theta_l}^{(l)}(\cdot)$ with a set of parameters θ_l. Usually, such a model is a neural network. Here, $f_{\theta_1}^{(1)}$ is called the first layer, $f_{\theta_2}^{(2)}$ the second layer, and so forth. Due to this stack of functions, a deep neural network can build its own features from an input. Hence, it suffices to use raw inputs, e.g., the pixel values of an $n \times m$ image, instead of manually creating features. For example, in image recognition applications, such handcrafted features could be the distance between the eyes, the roundness of the face, the nose width, or the cheekbone structure of a human face [BP93].

Graph Neural Networks One caveat of the mentioned neural network architectures, except for RNNs, is that they can only handle inputs of equal dimensions. For example, if a CNN is build for 32×32 pixel images, it can only operate on inputs of this dimension. Another limitation of these architectures is that they are not able to reflect the underlying structure of an object or problem. Therefore, researchers have come up with an architecture called GNNs. GNNs solve the discussed issues by using a graph to model the structure of a problem and assigning feature vectors to the nodes and edges of this graph. In this overview on GNNs, we follow the definitions of Wu et al. [WPC+20]. As we have already discussed in Section 3.2, there are four groups of GNNs: RecGNNs, ConvGNNs, STGNNs, and GAEs. The most used classes in the domain of combinatorial optimization, which we consider in this thesis, are the RecGNN and ConvGNN. Since the difference of those two groups lies in the fact that RecGNNs use the same set of parameters across all layers and the ConvGNNs has a different parameter set in each layer, we focus on ConvGNNs here. Similar to classical CNNs, also ConvGNNs follow the concept of convolutions. The idea behind convolutions is to aggregate the information stored in a node and all its neighboring nodes. In Figures 4.3a and 4.3b, we show the differences of 2-dimensional convolutions such as applied in CNNs and graph convolutions. The goal of GNNs is to create a representation of a node by iteratively updating its current features with graph convolution applications.

The class of ConvGNNs contains several different models, such as the MPNN model, which is the most generic ConvGNN due to its structure. We here present the MPNN to provide a more detailed overview on how GNNs work. Let $G = (V, E)$ be a graph, where V is the set of vertices or nodes with $n = |V|$, and E the set of edges with $m = |E|$. Let a node be denoted by $v \in V$, and an edge pointing from v to u by $(v, u) \in E$. The neighborhood of a node v is defined by $\mathcal{N}(v) = \{u \in V \mid (v, u) \in E\}$. Furthermore, in the considered context, the graph is an attributed graph, i.e., it has node attributes given by matrix $X \in \mathbb{R}^{n \times d}$, where $X_v \in \mathbb{R}^d$ is the feature vector of a node v. Additionally, also the edges may have attributes. These are represented by an edge feature matrix $X^e \in \mathbb{R}^{m \times c}$, where $X_{v,u}^e \in \mathbb{R}^c$ is the feature vector of an edge (v, u). To update the representation $H_v^{(k)}$ of a node v in layer k, the MPNN applies the following message-passing update



(a) 2-dimensional convolution: Each node represents a pixel from an image. Assuming the fully red node is chosen, a 2-D convolution selects the neighbors of the red node and takes an weighted aggregation of their values. Here, the neighborhood is of fixed size determined by a filter size. Figure recreated from Wu et al. [WPC+20].



(b) Graph convolution: The neighbors of a selected node (fully red node) are defined by the structure of the graph. The graph convolution takes a weighted aggregation over these neighbors. The size of the neighborhood is flexible and depends on the selected node. Figure recreated from Wu et al. [WPC+20].

Figure 4.3: Visual representations of a 2-dimenstional and a graph convolution.

function

$$H_v^{(k+1)} = U_k\left(H_v^{(k)}, \sum_{u \in N(v)} M_k(H_v^{(k)}, H_u^{(k)}, X_{v,u}^e)\right) \tag{4.6}$$

where U_k and M_k usually are parameterized, learnable functions, and $H^{(0)} = X$. Assume that the applied MPNN consists of K layers in total. In the beginning, $H_v^{(0)} = X_v$ for each node $v \in V$. To update the node representation $H_v^{(0)}$ of an arbitrary node $v \in V$ and transform it into $H_v^{(1)}$, the MPNN applies update function (4.6). In this update function, the neural network first aggregates the information contained in v's neighbors $u \in N(v)$. Therefore, the learnable function M_0 takes the current node vector $H_v^{(0)}$, the current representation $H_u^{(0)}$ of a neighbor u, and the associated edge feature vector $X_{v,u}^e$ as inputs and computes a so-called message. The genericity of the MPNN lies in the fact that M_0 can be any learnable function. For example, it could be an MLP taking the concatenation of $H_v^{(0)}$, $H_u^{(0)}$, and $X_{v,u}^e$ as an input. It could also be a simplified function where some of the inputs are not required. Next, $m_v^{(0)} = \sum_{u \in N(v)} M_0(H_v^{(0)}, H_u^{(0)}, X_{v,u}^e)$ represents the final neighborhood aggregation for node v, where the messages obtained from all neighbors of v are summed up into a single message $m_v^{(0)}$. Finally, the second function U_0 of the first layer is applied on the current node representation $H_v^{(0)}$ and message $m_v^{(0)}$. It updates v's representation $H_v^{(1)} = U_0(H_v^{(0)}, m_v^{(0)})$. Again, U_0 can be any learnable function. The MPNN performs the described node update for each node $v \in V$ to compute its representation for the next layer. Once every node representation is updated, the whole process is repeated until the final output representation $H_v^{(K)}$ for each node $v \in V$ is obtained. The size of these final node vectors depends on the structure of the MPNN.

These computed node representations can be used for different tasks such as node or graph classification. In node classification, an MLP is usually independently applied on each final node representation vector to classify this node. In graph classification, the final node vectors are aggregated by taking the sum or mean, and then an MLP is employed on this single vector to classify the whole graph.

CHAPTER 5

Problem Formalization

In this section, we formalize the SRRP stated in Section 2. First, we establish the needed notation for the formalization. Then, we apply this notation to define MILPs for the classical staff rostering problem and the SRRP.

5.1 Notation

A summary of the used notation can be found in Table 5.1. The SRRP is defined on a set of employees $N = \{1, \ldots, k\}$, a set of days representing the time horizon $D = \{1, \ldots, h\}$, and a set of shifts $S = \{\mathtt{E}, \mathtt{D}, \mathtt{N}, \mathtt{F}\}$, where `E` stands for the early shift, which lasts from 7 am to 3 pm, `D` represents the day shift lasting from 3 pm to 11 pm, and `N` denotes the night shift starting at 11 pm and ending at 7 am. Lastly, `F` stands for the free shift where an employee does not have to work. Furthermore, $R_{ds} \in \mathbb{N}_0$ holds the number of employees needed for each shift $s \in S$ on each day $d \in D$. This corresponds to the formalization of the staffing requirements. Moreover, each employee $n \in N$ states a preference for each shift $s \in S$ of each day $d \in D$ expressed by the value $P_{nds} \in \{1, \ldots, |S|\}$. The lower the number, the more desirable it is for an employee to do a shift.

To account for the minimum and maximum constraints HWSA, HCWSA, HSTA, and HCSTA from Section 2, the following parameters are given. Parameters $\alpha^{\min}, \alpha^{\max} \in \mathbb{N}_0$ represent the minimum and maximum number of working assignments in the planning horizon, respectively. Therefore, they refer to constraint HWSA stating that an employee must not be assigned to less than a minimum or more than a maximum number of working shifts in the scheduling period. HCWSA concerns the minimum and maximum number of allowed consecutive working days. These minimum and maximum numbers are denoted by $\beta^{\min}, \beta^{\max} \in \mathbb{N}_0$. Parameters $\gamma_s^{\min}, \gamma_s^{\max} \in \mathbb{N}_0$ are the minimum and maximum number of assignments per shift $s \in S$ in the planning horizon. They refer to constraint HSTA regulating the allowed number of assignments to a specific shift type. Finally, parameters $\delta_s^{\min}, \delta_s^{\max} \in \mathbb{N}_0$ represent the minimum and maximum number

of consecutive working assignments per shift $s \in S$. Hence, they belong to constraint HCSTA stating the permitted number of consecutive assignments to a shift type.

So far, the inputs are similar to the classical NRP. The SRRP requires the following additional information. To compute the changes made to the original schedule. The old roster prior to the disruptions is represented in the following way. The value $x^{\text{old}}_{nds} = 1$ if employee $n \in N$ is scheduled for shift $s \in S$ on day $d \in D$ in the original roster and $x^{\text{old}}_{nds} = 0$ otherwise. The disruptions are modeled as follows. First, we distinguish between single- and multi-shift disruptions. The single-shift disruptions are represented by a set of triples U^{ssd}, where each triple $(n, d, s) \in U^{\text{ssd}}$ indicates that employee $n \in N$ is unable to cover shift $s \in S$ on day $d \in D$. For multi-shift disruptions, we consider whole days on which an employee is unable to work at all. Thus, $U^{\text{msd}}_{nd} = 1$ if and only if employee $n \in N$ cannot work on day $d \in D$ and $U^{\text{msd}}_{nd} = 0$ otherwise. Second, $R^{\text{c}}_{ds} \in \mathbb{Z}$ shows how the demand changed for each shift $s \in S$ on each day $d \in D$ compared to the original demand.

The decision variables used to solve the SRRP are the following:

$$x_{nds} = \begin{cases} 1 & \text{if employee } n \in N \text{ is scheduled to work shift } s \in S \text{ on day } d \in D \\ 0 & \text{otherwise} \end{cases}$$

To compute overstaffing and understaffing, we make use of auxiliary variables. Variables $v^{\text{SCREQ-}}_{ds} \in \mathbb{N}_0$ state by how many employees shift $s \in S$ on day $d \in D$ is overstaffed. In other words, $v^{\text{SCREQ-}}_{ds}$ indicates how many employees would have to be withdrawn from shift s on day d to cover the staffing requirements exactly. Complementary, $v^{\text{SCREQ+}}_{ds} \in \mathbb{N}_0$ holds the same information for a potential understaffing of shift $s \in S$ on day $d \in D$. We explicitly introduce two variables for overstaffing and understaffing as this enables us to define a different penalty weight for each of them. All the weights are represented by $\omega^i \in \mathbb{N}_0$ where i stands for the type of constraints. Weight $\omega^{\text{SCREQ-}}$ refers to the penalization for overstaffing and $\omega^{\text{SCREQ+}}$ for understaffing. The penalty term of a new schedule regarding coverage requirement violations is then computed in the following way:

$$\sum_{d \in D} \sum_{s \in S} (\omega^{\text{SCREQ+}} \cdot v^{\text{SCREQ+}}_{ds} + \omega^{\text{SCREQ-}} \cdot v^{\text{SCREQ-}}_{ds}) \tag{5.1}$$

Further auxiliary variables are needed to compute how well employee preferences have been taken into account. The, variable $v^{\text{SPREF}}_{nds} \in \{0, \ldots, |S|\}$ is assigned the value P_{nds} if employee $n \in N$ is assigned to shift $s \in S$ on day $d \in D$ and $v^{\text{SPREF}}_{nds} = 0$ otherwise. The penalty term for employee preferences is defined in (5.2), where the associated weight value is ω^{SPREF}.

$$\sum_{n \in N} \sum_{d \in D} \sum_{s \in S} \omega^{\text{SPREF}} \cdot v^{\text{SPREF}}_{nds} \tag{5.2}$$

Next, the variables $v^{\text{SEWL-}}_{n} \in \mathbb{R}_{\geq 0}$ and $v^{\text{SEWL+}}_{n} \in \mathbb{R}_{\geq 0}$ are assigned the numbers of working assignments that employee $n \in N$ works more or less than the average employee,

Table 5.1: Notation for the staff rostering problem and the SRRP formulation.

Symbol	Definition
Input Data	
$n \in N$	n is the index of the employee, and N is the set of employees
$d \in D$	d is the index of the day and D is the set of days
$s \in S$	s denotes a shift from the set of shifts $S = \{\mathtt{E}, \mathtt{D}, \mathtt{N}, \mathtt{F}\}$
$R_{ds} \in \mathbb{N}_0$	number of required employees for shift s on day d
$R^{\mathrm{c}}_{ds} \in \mathbb{Z}$	change in demand for shift s on day d compared to the original roster
$P_{nds} \in \{1, \ldots, \lvert S\rvert\}$	the lower the number, the more desirable for employee n to work shift s on day d
$x^{\mathrm{old}}_{nds} \in \{0, 1\}$	1 if employee n is scheduled for shift s on day d in the original roster, otherwise 0
$(n, d, s) \in U^{\mathrm{ssd}}$	set of single-shift disruptions (n, d, s) indicating that employee n is unable to cover shift s on day d
$U^{\mathrm{msd}}_{nd} \in \{0, 1\}$	multi-shift disruptions denoting that employee $n \in N$ is unable to work on day $d \in D$
$(s1, s2) \in S^{-}$	an assignment to shift s_1 on day d must not be followed by an assignment to shift s_2 on day $d + 1$
$\alpha^{\min}, \alpha^{\max} \in \mathbb{N}_0$	minimum and maximum number of working days in planning horizon
$\beta^{\min}, \beta^{\max} \in \mathbb{N}_0$	minimum and maximum number of consecutive working days
$\gamma^{\min}_s, \gamma^{\max}_s \in \mathbb{N}_0$	minimum and maximum number of assignments for shift s in the planning horizon
$\delta^{\min}_s, \delta^{\max}_s \in \mathbb{N}_0$	minimum and maximum number of consecutive assignments to shift s
$\omega^{\text{SCREQ-}} \in \mathbb{N}_0$	weight for overstaffing violations
$\omega^{\text{SCREQ+}} \in \mathbb{N}_0$	weight for understaffing violations
$\omega^{\text{SPREF}} \in \mathbb{N}_0$	weight for preference violations
$\omega^{\text{SEWL}} \in \mathbb{N}_0$	weight for uneven workloads
$\omega^{\text{SMOD}} \in \mathbb{N}_0$	weight for changes in original schedule
Decision Variables	
$x_{nds} \in \{0, 1\}$	1 if employee n is scheduled for shift s on day d in the solution roster, otherwise 0
Auxiliary Variables	
$v^{\text{SCREQ-}}_{ds} \in \mathbb{N}_0$	overstaffing violations for shift s on day d
$v^{\text{SCREQ+}}_{ds} \in \mathbb{N}_0$	understaffing violations for shift s on day d
$v^{\text{SPREF}}_{nds} \in \{0, \ldots, \lvert S\rvert\}$	penalty cost of assigning employee n to shift s on day d
$v^{\text{SEWL-}}_{n} \in \mathbb{R}_{\geq 0}$	number of working assignments that employee n works more than the average employee
$v^{\text{SEWL+}}_{n} \in \mathbb{R}_{\geq 0}$	number of working assignments that employee n works less than the average employee
$v^{\text{SMOD}}_{nds} \in \{0, 1\}$	1 if x_{nds} is different from x^{old}_{nds} for an employee n and shift s on day d, otherwise 0

respectively. Weight ω^{SEWL} is used to penalize this potential unevenness in the workload. The total associated penalty value is

$$\sum_{n \in N} \omega^{\text{SEWL}} \cdot (v_n^{\text{SEWL+}} + v_n^{\text{SEWL-}}). \tag{5.3}$$

Additionally, ω^{SMOD} denotes the penalty weight for each modification made to the old schedule. To count these modifications, we introduce auxiliary variables $v_{nds}^{\text{SMOD}} \in \{0, 1\}$. For employee $n \in N$ on day $d \in D$ and shift $s \in S$, variable $v_{nds}^{\text{SMOD}} = 1$ if and only if x_{nds} is different from x_{nds}^{old}. The penalty term for penalizing violations changes to the original roster is defined in (5.4). The value is divided by two since each assignment change leads to two modifications.

$$\frac{1}{2} \sum_{n \in N} \sum_{d \in D} \sum_{s \in S} \omega^{\text{SMOD}} \cdot v_{nds}^{\text{SMOD}} \tag{5.4}$$

Both the objective functions of the classical staff rostering problem and the SRRP are based on these penalty terms. For the classical problem, the goal is to minimize the penalties regarding the staffing requirements (SCREQ), the preferences (SPREF), and the unevenness in the workload (SEWL). Therefore, its objective function is defined as

$$\min \quad (5.1) + (5.2) + (5.3). \tag{5.5}$$

The SRRP additionally needs to account for the modifications compared to the original roster (SMOD). Since the goal to have as few changes as possible to the old schedule implicitly covers the preference and even workload requirements, we omit those penalty terms in the objective function of the SRRP and state it as

$$\min \quad (5.1) + (5.4). \tag{5.6}$$

In Section 6.2, we formulate a MILP, for which we relax the hard constraints and turn them into soft constraints. Therefore, additional auxiliary variables and weights are required. For better readability, we state and discuss them in the respective section.

5.2 Mathematical Formulations

In the following, we apply the introduced notation to define two MILPs, one for the classical staff rostering problem and one for the SRRP. We distinguish between these two models since we require a separate program for creating the original schedules in the test instance generation (see Section 8.1). Furthermore, the model of the regular rostering problem builds the basis for the SRRP program. Hence, we can directly reuse some of the constraint definitions and objectives for the SRRP. Finally, we introduce an additional MILP model with some of the hard constraints transformed into soft constraints.

5.2.1 Staff Rostering Problem Model

We now formulate the mixed-integer linear program for the classical staff rostering problem including the constraints defined in Section 2.

$$\min \quad (5.1) + (5.2) + (5.3) \tag{5.7}$$

$$\text{s.t.} \quad \sum_{s \in S} x_{nds} = 1 \qquad \forall n \in N, d \in D \tag{5.8}$$

$$x_{nds_1} + x_{n(d+1)s_2} \leq 1 \qquad \forall n \in N, (s_1, s_2) \in S^-, d \in \{1, \dots, |D| - 1\} \tag{5.9}$$

$$\alpha^{\min} \leq \sum_{d \in D} \sum_{s \in S \setminus \{\mathtt{F}\}} x_{nds} \leq \alpha^{\max} \qquad \forall n \in N \tag{5.10}$$

$$\sum_{s \in S \setminus \{\mathtt{F}\}} \sum_{d'=d}^{\min(|D|, d+\beta^{\min}-1)} x_{nd's} \geq \beta^{\min} \cdot \left(\sum_{s \in S \setminus \{\mathtt{F}\}} (x_{nds} - x_{n(d-1)s}) \right) \qquad \forall n \in N, d \in D \tag{5.11}$$

$$\sum_{s \in S \setminus \{\mathtt{F}\}} \sum_{d'=d}^{d+\beta^{\max}} x_{nd's} \leq \beta^{\max} \qquad \forall n \in N, d \in \{1, \dots, |D| - \beta^{\max}\} \tag{5.12}$$

$$\gamma_s^{\min} \leq \sum_{d \in D} x_{nds} \leq \gamma_s^{\max} \qquad \forall n \in N, s \in S \tag{5.13}$$

$$\sum_{d'=d}^{\min(|D|, d+\delta_s^{\min}-1)} x_{nd's} \geq \delta_s^{\min} \cdot (x_{nds} - x_{n(d-1)s}) \qquad \forall n \in N, d \in D, s \in S \tag{5.14}$$

$$\sum_{d'=d}^{d+\delta_s^{\max}} x_{nd's} \leq \delta_s^{\max} \qquad \forall n \in N, s \in S, d \in \{1, \dots, |D| - \delta_s^{\max}\} \tag{5.15}$$

$$\sum_{n \in N} x_{nds} + v_{ds}^{\text{SCREQ+}} - v_{ds}^{\text{SCREQ-}} = R_{ds} \qquad \forall d \in D, s \in S \tag{5.16}$$

$$v_{nds}^{\text{SPREF}} = P_{nds} \cdot x_{nds} \qquad \forall n \in N, d \in D, s \in S \tag{5.17}$$

$$\sum_{d \in D} \sum_{s \in S \setminus \{\mathtt{F}\}} x_{nds} + v_n^{\text{SEWL+}} - v_n^{\text{SEWL-}} = \frac{1}{|N|} \sum_{n' \in N} \sum_{d \in D} \sum_{s \in S \setminus \{\mathtt{F}\}} x_{n'ds} \qquad \forall n \in N \tag{5.18}$$

$$x_{nds} \in \{0, 1\} \qquad \forall n \in N, d \in D, s \in S \tag{5.19}$$

$$v_n^{\text{SCREQ+}}, v_n^{\text{SCREQ-}} \geq 0 \qquad \forall d \in D, s \in S \tag{5.20}$$

$$0 \leq v_{nds}^{\text{SPREF}} \leq |S| \qquad \forall n \in N, d \in D, s \in S \tag{5.21}$$

$$v_n^{\text{SEWL+}}, v_n^{\text{SEWL-}} \geq 0 \qquad \forall n \in N \tag{5.22}$$

$$x_{n0s} = 0 \qquad \forall n \in N, s \in S \tag{5.23}$$

Constraints (5.8) ensure that an employee is assigned exactly to one shift, including the free shift, on each day (HOSPD). Together with Constraints (5.8), Constraints (5.9) enforce the mandatory rest of at least eleven hours between two working shifts (HREST). This is realized by disallowing the following working shift sequences for two consecutive days: night to early shift, night to day shift and day to early shift. These invalid sequences are stored as tuples in $S^- = \{(\texttt{N}, \texttt{E}), (\texttt{N}, \texttt{D}), (\texttt{D}, \texttt{E})\}$. Hard constraint HWSA states that an employee must be assigned to a number of working shifts within a minimum ($\alpha^{\min}$) and a maximum ($\alpha^{\max}$) in the planning horizon. This constraint is considered in Constraints (5.10). Constraints (5.11) and (5.12) enforce that each employee works at least $\beta^{\min}$ and at most $\beta^{\max}$ consecutive days, respectively. They refer to hard constraint HCWSA. The realization of Constraints (5.12) is straight forward since for each $\beta^{\max} + 1$ consecutive days and each employee there must be at most $\beta^{\max}$ working assignments. Constraints (5.11), however, are slightly more complex. There, $\beta^{\min}$ only has to be enforced as a lower bound if employee n is assigned to work on day d but is assigned a free shift the day before d. According to Constraints (5.13), each employee must have a minimum of $\gamma_s^{\min}$ and a maximum of $\gamma_s^{\max}$ assignments to a shift $s \in S$ in the whole scheduling period (HSTA). Constraints (5.14) and (5.15) are similar to Constraints (5.11) and (5.12) but they ensure that each employee has at least $\gamma_s^{\min}$ and at most $\gamma_s^{\max}$ consecutive assingments to a certain shift $s \in S$ (HCSTA).

So far, we have discussed the formalizations of the hard constraints in the model. Now, we deal with the soft constraints. More specifically, we show how to compute the values for the auxiliary variables associated with the soft constraints. These are then used to calculate the penalties, and therefore the objective value of a solution. According to Constraints (5.16) the demand on each day for each shift should be satisfied (SCREQ). However, the sum of employee assignments does not necessarily match the demand exactly. Instead, auxiliary variables $v_{ds}^{\text{SCREQ-}}$ and $v_{ds}^{\text{SCREQ+}}$ compensate the case when too many or not enough employees are assigned to shift s on day d, respectively.

To incorporate employee preferences as required by soft constraint (SPREF), Constraints (5.17) are introduced. Finally, Equations (5.18) are defined to help calculate the equal workload violations from soft constraint SEWL. The right-hand side of the equation computes the average number of assignments to working shifts for all employees in the planning period. Then, c_n is used to determine the difference between this average and an employee n's total number of working shifts.

The goal of the staff rostering problem is to create a high-quality employee duty schedule. More formally, high-quality means that all hard constraints are satisfied and the soft constraints are fulfilled as well as possible. We have already introduced the auxiliary variables $v_{ds}^{\text{SCREQ+}}$, $v_{ds}^{\text{SCREQ-}}$, v_{nds}^{SPREF}, $v_n^{\text{SEWL+}}$, and $v_n^{\text{SEWL-}}$ representing the violations of soft constraints. Their corresponding penalty terms (5.1), (5.2), and (5.3) are used in the objective function (5.7) that should be minimized. Again, (5.1) calculates the penalty for the overstaffing and understaffing violations (SCREQ), (5.2) represents the preference violations (SPREF), and (5.3) penalizes unequal workloads among employees (SEWL). For completeness, we state the domains of the model's variables in Inequalities (5.19)

- (5.22) and Constraints (5.23). As it is possible in Constraints (5.11) and (5.14) that $d-1=0 \notin D$, we add constants $x_{n0s}=0$ for $n \in N$, $s \in S$, to account for this fact and to keep the constraints valid. All these domain specifications hold for the rerostering problem as well.

5.2.2 Staff Rerostering Problem Model

In this section, we state the mixed-integer linear program for the SRRP. Therefore, we reuse some of the previously defined constraints and combine them with new SRRP-specific constraints.

$$\min \quad (5.1)+(5.4) \tag{5.24}$$

$$\text{s.t.} \quad (5.8),(5.9),(5.11),(5.12),(5.14),(5.19) \text{ to } (5.23)$$

$$\alpha^{\min} - \sum_{d\in D} U^{\text{msd}}_{nd} \leq \sum_{d\in D}\sum_{s\in S\setminus\{\text{F}\}} x_{nds} \leq \alpha^{\max} \qquad \forall n \in N \tag{5.25}$$

$$\gamma^{\min}_s - \sum_{d\in D} U^{\text{msd}}_{nd} \leq \sum_{d\in D} x_{nds} \leq \gamma^{\max}_s \qquad \forall n \in N, s \in S\setminus\{\text{F}\} \tag{5.26}$$

$$\gamma^{\min}_{\text{F}} \leq \sum_{d\in D} x_{nd\text{F}} \leq \gamma^{\max}_{\text{F}} + \sum_{d\in D} U^{\text{msd}}_{nd} \qquad \forall n \in N \tag{5.27}$$

$$\sum_{d'=d}^{d+\delta^{\max}_s} x_{nd's} \leq \delta^{\max}_s \qquad \forall n \in N, s \in S\setminus\{\text{F}\}, d \in \{1,\dots,|D|-\delta^{\max}_s\} \tag{5.28}$$

$$\sum_{d'=d}^{d+\delta^{\max}_{\text{F}}} x_{nd'\text{F}} \leq \delta^{\max}_{\text{F}} + \sum_{d'=d}^{d+\delta^{\max}_{\text{F}}} U^{\text{msd}}_{nd'} \qquad \forall n \in N, d \in \{1,\dots,|D|-\delta^{\max}_{\text{F}}\} \tag{5.29}$$

$$\sum_{s\in S\setminus\{\text{F}\}} x_{nds} \leq 1 - U^{\text{msd}}_{nd} \qquad \forall n \in N, d \in D \tag{5.30}$$

$$x_{nds} = 0 \qquad \forall (n,d,s) \in U^{\text{ssd}} \tag{5.31}$$

$$\sum_{n\in N} x_{nds} + v^{\text{SCREQ+}}_{ds} - v^{\text{SCREQ-}}_{ds} = R_{ds} + R^{\text{c}}_{ds} \qquad \forall d \in D, s \in S \tag{5.32}$$

$$v^{\text{SMOD}}_{nds} \leq x^{\text{old}}_{nds} + x_{nds} \qquad \forall n \in N, d \in D, s \in S \tag{5.33}$$

$$v^{\text{SMOD}}_{nds} \geq x^{\text{old}}_{nds} - x_{nds} \qquad \forall n \in N, d \in D, s \in S \tag{5.34}$$

$$v^{\text{SMOD}}_{nds} \geq x_{nds} - x^{\text{old}}_{nds} \qquad \forall n \in N, d \in D, s \in S \tag{5.35}$$

$$v^{\text{SMOD}}_{nds} \leq 2 - x_{nds} - x^{\text{old}}_{nds} \qquad \forall n \in N, d \in D, s \in S \tag{5.36}$$

$$v^{\text{SMOD}}_{nds} \in \{0,1\} \qquad \forall n \in N, d \in D, s \in S \tag{5.37}$$

In the previous section, we defined Constraints (5.8)–(5.15) that formalize the hard constraints HOSPD, HREST, HWSA, HCWSA, HSTA, and HCSTA. From these, (5.8), (5.9), (5.11), (5.12), and (5.14) can be directly reused for the MILP model of the SRRP.

However, the formalization of hard constraints HWSA, HSTA, and HCSTA requires adaption due to the multi-shift disruptions. Since these disruptions are mostly caused by illness and vacation, an employee could be unavailable for multiple consecutive days. Therefore, it might be impossible to adhere to the minimum number of working shifts, the maximum number of free shifts, and the maximum number of consecutive free shifts constraints. Thus, Constraints (5.25)–(5.29) are introduced to compensate for this fact. The main idea behind these adapted variants is to incorporate the days of absence to reduce the lower bounds or raise the upper bounds. Additionally, Constraints (5.30) and Constraints (5.31) refer to hard constraint HABSE and ensure that an employee cannot be assigned to a shift the employee is unable to cover.

In contrast to the hard constraints, we do not consider all the soft constraints from the previous section for the rerostering model. That is the case for constraints SPREF and SEWL. They deal with the requirements that employee preferences should be included in the schedule and that the roster should have an even workload for each employee, respectively. The reason we do not include them in the rerostering formulation is that they are implicitly covered by the goal to be as close as possible to the original solution (SMOD). For the previously formalized staffing constraints SCREQ, adopted formulations have to be introduced due to the disruptions. Therefore, Constraints (5.32) ensure that the staffing requirements for each shift are covered. They extend the cover Constraint (5.16) from the last section by adding R^{c}_{ds} to the right-hand side of the equation. Hence, a change of demand, which can be caused by track work or train failures, is included in the model. Finally, constraint SMOD is SRRP-specific and states that there should be as few changes as possible to the original roster. Constraints (5.33)–(5.36) model these changes from the original schedule to the new roster incorporating the disruptions. We achieve this by combining the auxiliary variable v^{SMOD}_{nds} with exclusive-OR constraints. This means that $v^{\text{SMOD}}_{nds} = 1$ if and only if the values of x^{old}_{nds} and x_{nds} are different for an employee $n \in N$, a shift $s \in S$, and a day $d \in D$. Hence, either $x^{\text{old}}_{nds} = 1$ or $x_{nds} = 1$ but not both. Again, using these constraints may very likely lead to some penalties that cannot be avoided but have no consequences when solving the problem.

Finally, the objective function (5.24) of the SRRP consists of two parts. The first one, (5.1), refers to the violations of the coverage requirements constraint (SCREQ). The second one, (5.4), corresponds to the penalties for changes in the schedule compared to the old roster (SMOD). In (5.4), the value is divided by two to indicate that every modification is counted twice by Constraints (5.33)–(5.36). As previously mentioned, the domain specifications of the staff rostering problem can be reused as well. Additionally, the domain for the rerostering-specific variables v^{SMOD}_{nds} is stated in Constraints (5.37).

CHAPTER 6

Large Neighborhood Search

We have already pointed out that Moz and Pato [MP07] proved the NRRP, which is a simpler variant of the SRRP, to be NP-hard. Hence, solely relying on exact solutions methods is not a promising approach to efficiently solve large instances of the SRRP. Therefore, we propose a LNS meta-heuristics to find high-quality solutions fast for this hard problem. For a general review of the LNS, we refer to Section 4.2.3. In this section, we introduce the LNS framework that is the foundation of our work. We define the construction heuristic and the repair operator used in each LNS applied in this thesis. Additionally, we propose a randomized destroy operator, serving as a benchmark in our computational experiments presented in Section 8.2. In the following section, we eventually define our learning-based destroy operator, representing the main contribution of this thesis.

Algorithm 4 shows the LNS pseudocode matching the requirements regarding our solution approaches. Since we use two alternative destroy methods in this work, our LNS-Framework takes a destroy operator `DestroySolution` in addition to an SRRP instance I as an input. We assume that an instance contains all the required input data to define the SRRP. For an extensive overview of the notation and data contained by an SRRP instance, we refer to the problem formalization in Section 5 and especially to Table 5.1. Each solution in the solution space $\mathcal{S}$ is represented by decision variables $x_{nds} \in \{0, 1\}$ for all employees $n \in N$, days $d \in D$, and shifts $s \in S$. If an employee n is scheduled for shift s on day d in the solution, $x_{nds} = 1$, otherwise $x_{nds} = 0$. Algorithm 4 starts by constructing an initial solution. We describe the construction heuristic `ConstructSolution` used to create those initial solutions for both LNS approaches in Section 6.1. Then, we enter the main optimization loop, where a solution first is partially destroyed. We review the classical randomized destroy method in Section 6.3 and afterward discuss the learning-based approach concerning the neural network architecture and the learning method used in Section 7. In the next step, we repair the partially destroyed solution. We distinguish between feasible and infeasible solutions in the repair operator and adapt our repair

Algorithm 4: LNS-Framework

Input: SRRP instance I, destroy method `DestroySolution`.

1 $x_{nds} \leftarrow$ `ConstructSolution`(I)
2 $x_{nds}^{\text{best}} \leftarrow x_{nds}$
3 **while** *termination criterion not met* **do**
4 feasible $\leftarrow$ `IsFeasible`(x_{nds})
5 $x_{nds}^{\text{partial}} \leftarrow$ `DestroySolution`(I, x_{nds})
6 $x_{nds} \leftarrow$ `RepairSolution`$(I, x_{nds}^{\text{partial}}, \text{feasible})$
7 **if** $c(x_{nds}) < c(x_{nds}^{best})$ **then**
8 $x_{nds}^{\text{best}} \leftarrow x_{nds}$
9 **else**
10 $x_{nds} \leftarrow x_{nds}^{\text{best}}$
11 **end**
12 **end**
13 **return** x_{nds}

mechanism accordingly. Therefore, we use `IsFeasible` to compute whether or not a solution is feasible. The repair method `RepairSolution` is discussed in Section 6.2. We only accept a new solution if it has been improved in the last destroy and repair cycle. Here, $c : \mathcal{S} \to \mathbb{R}$ represents the objective function. Eventually, we repeat the training loop until the termination criterion is met.

6.1 Construction Heuristic

Usually, the goal of construction heuristics is to create a promising feasible solution. However, obtaining feasible solutions is not always computationally easy, which also holds for the SRRP. Therefore, we suggest a fast construction heuristic, which generates solutions that may be infeasible but possibly can be transformed into high-quality feasible solutions quickly in the following LNS iterations. The idea is to take the original schedule before the disruptions as a solution and change employees' working assignments to non-working or free shifts if they are absent on a day or shift. Hence, we build on the fact that these former schedules were already optimized to minimize cover requirement violations. Additionally, this new solution has a low number of modifications compared to the old roster since we only change shifts if employees are absent and can therefore be considered a promising starting point.

Algorithm 5 shows the pseudocode for the construction heuristic `ConstructSolution`. In Lines 1-3, we first retrieve problem-specific information such as the sets of employees N, days D, and shifts S from the instance. Moreover, we obtain the multi-shift disruptions $U_{nd}^{\text{msd}} \in \{0, 1\}$ for each employee $n \in N$ and day $d \in D$. Here, $U_{nd}^{\text{msd}} = 1$, if an employee n is absent for all shifts on a day d, otherwise $U_{nd}^{\text{msd}} = 0$. Lastly, we collect U^{ssd} representing the set of single-shift disruptions, which contains tuples (n, d, s) for each employee n

Algorithm 5: ConstructSolution

Input: SRRP instance I.

$N, D, S \leftarrow$ `GetEmployeesDaysShifts`(I)
$U^{\text{msd}}_{nd}, U^{\text{ssd}} \leftarrow$ `GetDisruptions`(I) // multi- and single-shift disruptions
$x^{\text{old}}_{nds} \leftarrow$ `GetOldSchedule`(I) // original schedule prior to the disruptions
$x_{nds} \leftarrow x^{\text{old}}_{nds} \quad \forall n \in N, d \in D, s \in S$
for $n \in N$ **do**
  for $d \in D$ **do**
    if $(\exists s \in S : (n, d, s) \in U^{ssd})$ *or* $(U^{msd}_{nd} == 1)$ **then**
      $x_{nds} \leftarrow 0 \quad \forall s \in S \setminus \{\texttt{F}\}$
      $x_{nd\texttt{F}} \leftarrow 1$ // assign employee n to free shift on day d
    end
  end
end
return x_{nds}

that is absent for a shift s on a day d. The algorithm then works by copying the values from the old roster x^{old}_{nds} to the initial solution x_{nds} for all $n \in N$, $d \in D$, and $s \in S$. Then, it checks for all employees $n \in N$ and days $d \in D$ whether the employee is absent from the previously assigned shift or the whole day d. If this is the case, the employee is withdrawn from this previous shift and assigned to the free shift.

6.2 Repair Operator

As we have discussed in the previous section, our construction heuristic may return infeasible solutions. Additionally, the destroy operator might not select all the relevant variables required to turn the solution feasible with one repair operator application. As a consequence, we have to deal with infeasible solutions during the repair operation and the LNS in general. Therefore, we propose a further MILP model, where we transform some of the hard constraints into soft constraints. The relaxed hard constraints are HREST, HWSA, HSTA, HCWSA, and HCSTA. We recap the meaning of these constraints when introducing the auxiliary variables required for this relaxation. See also Section 2 for the definitions of these constraints. As a result, a solution to the new model can violate these hard constraints at the cost of additional penalizations. To reduce the chance that infeasible solutions have better objective values than any feasible solution, we assign these weights a value 100 times greater than the highest weight from the regular MILP. Using this model in the repair operator of the LNS allows us to start the search from an infeasible solution and gradually move towards a feasible one.

Before we formulate the MILP, we discuss our repair operator `RepairSolution` shown in Algorithm 6. As input, it takes an SRRP instance and a partial solution where a subset of decision variables was freed. The actual repairing of the solution lies in solving

Algorithm 6: RepairSolution

Input: SRRP instance I, partial solution x_{nds}^{partial}, boolean value which is true if and only if the solution was feasible before destruction ν.

if ν **then**
  $x_{nds} \leftarrow$ solve x_{nds}^{partial} with standard MILP from Section 5.2.2
else
  $x_{nds} \leftarrow$ solve x_{nds}^{partial} with MILP having hard constraints relaxed
end
return x_{nds}

the subproblem induced by fixing the other variables. Another essential input is the information on whether or not the partial solution was feasible before destruction. If it was, we apply the standard MILP from Section 5.2.2 to solve the mentioned subproblem since we know that the resulting solution will be feasible again. And if the solution was infeasible, we employ the MILP with relaxed hard constraints, which we state in the remainder of this section.

To transform hard constraints into soft constraints, we introduce the following auxiliary variables to account for hard constraint violations. Variables $v_{nds}^{\text{HREST}} \in \mathbb{N}_0$ for $n \in N$, $d \in D$, and $s \in S$ are established to identify violations of the minimum rest between work shifts, thereby refering to constraint HREST. Next, $v_{n}^{\text{HWSA+}} \in \mathbb{N}_0$ and $v_{n}^{\text{HWSA-}} \in \mathbb{N}_0$ indicate how many shifts an employee $n \in N$ has to work more or less, respectively, to lie within the allowed range of total working days in the planning horizon (HWSA). Similarly, $v_{ns}^{\text{HSTA+}} \in \mathbb{N}_0$ and $v_{ns}^{\text{HSTA-}} \in \mathbb{N}_0$ state how many days more or less an employee $n \in N$ must be assigned to a shift $s \in S$, such that the total number of assignments to shift s is within the allowed range for s (HSTA). To compute if the constraint for the minimum and the maximum number of consecutive working days is violated (HCWSA), variables $v_{nd}^{\text{HCWSA+}} \in \mathbb{N}_0$ and $v_{nd}^{\text{HCWSA-}} \in \mathbb{N}_0$ are introduced. For an employee $n \in N$ and a starting day $d \in D$, $v_{nd}^{\text{HCWSA+}}$ and $v_{nd}^{\text{HCWSA-}}$ represent the number of shifts that employee n has to work more or less in a specific horizon to satisfy HCWSA. Finally, $v_{nds}^{\text{HCSTA+}} \in \mathbb{N}_0$ and $v_{nds}^{\text{HCSTA-}} \in \mathbb{N}_0$ for $n \in N$, $d \in D$, and $s \in S$ similarly indicate violations of HCSTA, which defines the minimum and the maximum number of consecutive assignments to a shift s. The weights used to penalize the violations of hard constraints are given by $\omega^{\text{HREST}}, \omega^{\text{HWSA}}, \omega^{\text{HSTA}}, \omega^{\text{HCWSA}}$, and $\omega^{\text{HCSTA}} \in \mathbb{N}_0$.

$$
\begin{aligned}
\min \quad & \frac{1}{2} \sum_{n \in N} \sum_{d \in D} \sum_{s \in S} \omega^{\text{SMOD}} \cdot v_{nds}^{\text{SMOD}} \quad + \\
& \sum_{d \in D} \sum_{s \in S} (\omega^{\text{SCREQ+}} \cdot v_{ds}^{\text{SCREQ+}} + \omega^{\text{SCREQ-}} \cdot v_{ds}^{\text{SCREQ-}} \quad + \\
& \sum_{n \in N} \sum_{d \in D} \sum_{s \in S} \omega^{\text{HREST}} \cdot v_{nds}^{\text{HREST}} \quad +
\end{aligned}
$$

$$\sum_{n\in N} \omega^{\text{HWSA}} \cdot (v_n^{\text{HWSA+}} + v_n^{\text{HWSA-}})$$
$$\sum_{n\in N}\sum_{s\in S} \omega^{\text{HSTA}} \cdot (v_{ns}^{\text{HSTA+}} + v_{ns}^{\text{HSTA-}}) \quad +$$
$$\sum_{n\in N}\sum_{d\in D} \omega^{\text{HCWSA}} \cdot (v_{nd}^{\text{HCWSA+}} + v_{nd}^{\text{HCWSA-}}) \quad +$$
$$\sum_{n\in N}\sum_{d\in D}\sum_{s\in S} \omega^{\text{HCSTA}} \cdot (v_{nds}^{\text{HCSTA+}} + v_{nds}^{\text{HCSTA-}}) \tag{6.1}$$

s.t. (5.8), (5.19) to (5.23), (5.30) to (5.37)

$$x_{nds_1} + x_{n(d+1)s_2} - v_{nds_1}^{\text{HREST}} \leq 1$$
$$\forall n \in N, (s_1, s_2) \in S^-, d \in \{1, \ldots, |D| - 1\} \tag{6.2}$$

$$\alpha^{\min} - \sum_{d\in D} U_{nd}^{\text{msd}} \leq \sum_{d\in D}\sum_{s\in S\setminus\{\text{F}\}} x_{nds} + v_n^{\text{HWSA+}} - v_n^{\text{HWSA-}} \leq \alpha^{\max}$$
$$\forall n \in N \tag{6.3}$$

$$\sum_{s\in S\setminus\{\text{F}\}} \sum_{d'=d}^{\min(|D|, d+\beta^{\min}-1)} x_{nd's} + v_{nd}^{\text{HCWSA+}} \geq$$
$$\beta^{\min} \cdot \left(\sum_{s\in S\setminus\{\text{F}\}} (x_{nds} - x_{n(d-1)s}) \right)$$
$$\forall n \in N, d \in D \tag{6.4}$$

$$\sum_{s\in S\setminus\{\text{F}\}} \sum_{d'=d}^{d+\beta^{\max}} x_{nd's} - v_{nd}^{\text{HCWSA-}} \leq \beta^{\max} \qquad \forall n \in N, d \in \{1, \ldots, |D| - \beta^{\max}\} \tag{6.5}$$

$$\gamma_s^{\min} - \sum_{d\in D} U_{nd}^{\text{msd}} \leq \sum_{d\in D} x_{nds} + v_{ns}^{\text{HSTA+}} - v_{ns}^{\text{HSTA-}} \leq \gamma_s^{\max}$$
$$\forall n \in N, s \in S \setminus \{\text{F}\} \tag{6.6}$$

$$\gamma_{\text{F}}^{\min} \leq \sum_{d\in D} x_{nd\text{F}} + v_{n\text{F}}^{\text{HSTA+}} - v_{n\text{F}}^{\text{HSTA-}} \leq \gamma_{\text{F}}^{\max} + \sum_{d\in D} U_{nd}^{\text{msd}} \qquad \forall n \in N \tag{6.7}$$

$$\sum_{d'=d}^{\min(|D|, d+\delta_s^{\min}-1)} x_{nd's} + v_{nds}^{\text{HCSTA+}} \geq \delta_s^{\min} \cdot (x_{nds} - x_{n(d-1)s})$$
$$\forall n \in N, d \in D, s \in S \tag{6.8}$$

$$\sum_{d'=d}^{d+\delta_s^{\max}} x_{nd's} - v_{nds}^{\text{HCSTA-}} \leq \delta_s^{\max}$$
$$\forall n \in N, s \in S \setminus \{\text{F}\}, d \in \{1, \ldots, |D| - \delta_s^{\max}\} \tag{6.9}$$

$$\sum_{d'=d}^{d+\delta_F^{\max}} x_{nd'\text{F}} - v_{ndF}^{\text{HCSTA-}} \leq \delta_{\text{F}}^{\max} + \sum_{d'=d}^{d+\delta_{\text{F}}^{\max}} U_{nd'}^{\text{msd}}$$

$$\forall n \in N, d \in \{1, \ldots, |D| - \delta_{\mathrm{F}}^{\max}\} \quad (6.10)$$

$$v_n^{\text{HWSA+}}, v_n^{\text{HWSA-}} \geq 0 \qquad \forall n \in N \quad (6.11)$$

$$v_{ns}^{\text{HSTA+}}, v_{ns}^{\text{HSTA-}} \geq 0 \qquad \forall n \in N, s \in S \quad (6.12)$$

$$v_{nd}^{\text{HCWSA+}}, v_{nd}^{\text{HCWSA-}} \geq 0 \qquad \forall n \in N, d \in D \quad (6.13)$$

$$v_{nds}^{\text{HCSTA+}}, v_{nds}^{\text{HCSTA-}}, v_{nds}^{\text{HREST}} \geq 0 \qquad \forall n \in N, d \in D, s \in S \quad (6.14)$$

We do not relax hard constraints HOSPD and HABSE. They ensure that each employee is assigned only to one shift per day and that employees cannot be assigned to shifts when they are absent, respectively. Therefore, we include the regular formalizations of these constraints (5.8), (5.30), and (5.31) in the model. Constraints (5.19)–(5.23) and (5.32)–(5.37) refer to the soft constraints regarding the cover requirements and the modifications to the original roster, including the variable domain specifications. These are contained in the model as well. Furthermore, Constraints (6.2)–(6.9) represent the hard constraints modified into soft constraints. Compared to the original Constraints (5.8), (5.9), (5.11), (5.12), (5.14), and (5.25)–(5.29), each of the modified constraints contains an additional variable that is either added or subtracted. These variables are the variables described in the previous paragraph. They ensure that the lower and upper bounds of the minimum and the maximum constraints are respected. Therefore, they turn the hard into soft constraints. Constraints (6.11)–(6.14) define the domains of the new variables. Finally, the objective function of the SRRP with relaxed hard constraints is given by (6.1). It is similar to the objective function (5.24) of the regular SRRP with hard constraints. In addition to all the terms in objective function (5.24), objective function (6.1) contains the sums of the penalized hard constraints violations.

6.3 Randomized Destroy Operator

One intuitive approach to design a randomized destroy operator for the SRRP is to randomly sample an employee $n \in N$, a day $d \in D$, and a shift $s \in S$, destroy the respective variable x_{nds}, and repeat this process for some iterations. However, with this strategy, it is rather unlikely to sample multiple shifts for the same day and employee. Therefore, the later applied repair operator has frequently no or little chance to change the assignment of an employee and to improve the solution. To address this issue, a natural adaption of the mentioned destroy operator is to randomly select an employee $n \in N$, a day $d \in D$, and destroy the variables x_{nds} for all $s \in S$ instead of sampling the shifts too. Then, the repair operator has the guaranteed possibility to change the assignment of an employee on a specific day. Although this approach might seem promising, preliminary results have shown that it is not mature enough to obtain good results. The reason for this lies in the constraints regulating the consecutive number of working shifts (HCWSA) and the consecutive assignments per shift type (HCSTA). Here, the repair operator can

only produce improvements if consecutive days are selected, which is, similarly to the issue before, too improbable.

Algorithm 7: RandomDestroySolution

Input: SRRP instance I, current solution x_{nds}, number of destroy cycles z_1, number of days to destroy before and after selected day z_2.

```
1  N, D, S ← GetEmployeesDaysShifts(I)
2  employeeDayPairs ← {(n, d) | ∀n ∈ N, d ∈ D}
3  foreach i = 1, ..., z_1 do
4      (n, d) ← select random tuple from employeeDayPairs
5      period ← {max(1, d − z_2), ..., d, ..., min(|D|, d + z_2)}
6      foreach day d′ ∈ period do
7          remove (n, d′) from employeeDayPairs
8          destroy x_{nd′s} for all s ∈ S
9      end
10 end
11 return x_{nds}
```

A more elaborate strategy to solve the discussed challenges, is to destroy multiple consecutive days for an employee. Similar to the last variant, we randomly select an employee $n \in N$ and a day $d \in D$ forming an employee-day pair (n, d). Then, we pick the period $\{\max(1, d - z_2), \ldots, d, \ldots, \min(|D|, d + z_2)\}$ based on d, where we select the days ranging from z_2 days before d to z_2 days after d, respecting that one is the index of the first and $|D|$ the index of the last day. Afterward, we destroy $x_{nd's}$ for each shift $s \in S$, each day d' in the chosen period, and the selected employee n. In Algorithm 7, we show the `RandomDestroySolution` destroy method formulating this strategy. In `RandomDestroySolution`, the described process is repeated z_1 times. Both $z_1 \in \mathbb{N}_0$ and $z_2 \in \mathbb{N}_0$ are strategy parameters that Algorithm 7 needs as input. Note that we keep track of the already destroyed employee-day pairs (n, d') to reduce the number of overlaps. The upper bound of the total number of destructed employee-day pairs is $z_1 \cdot (2z_2 + 1)$.

When presenting the results in Section 8.2, we will show that the LNS utilizing the `RandomDestroySolution` destroy method delivers strong empirical performance. Moreover, we found that the insights obtained in this section are essential for efficiently solving the SRRP using an LNS. Therefore, we implement a similar consecutive-day selection scheme for our learning-based destroy operator introduced in the following section.

CHAPTER 7

Learning-Based Destroy Operator

The concept of our learning-based destroy operator is to utilize a neural network that, given an instance and a current solution represented by features, returns a policy to select promising employee-day pairs to destroy from $\{(n, d) \mid \forall n \in N, d \in D\}$. The objective is that the used ML model learns an intelligent selection policy through imitation learning, where we train the model to mimic an expert policy. Therefore, we pose the problem of selecting a destroy set as a MDP in Section 7.1. Our applied learning approach is based on conditional generative modeling, where the policy defines a distribution over the destroy sets. We establish this learning strategy in Section 7.2.

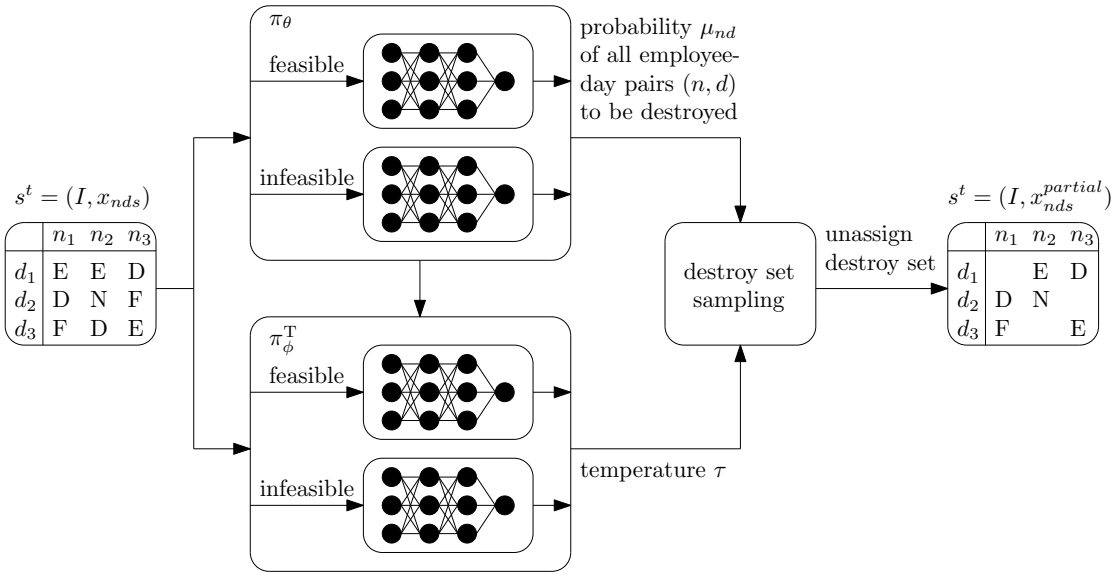


Figure 7.1: Overview of our learning-based destroy operator at test time.

Figure 7.1 provides an overview of our learning-based destroy operator. Here, π_θ represents the primary model. It takes a featurized version of a state $s^t \in ST$ at step t of an LNS run, consisting of an SRRP instance I and its current solution x^t_{nds}, as an input. As already stated in Section 6, we assume that an instance contains all the required input data to define the SRRP. We refer to the problem formalization in Section 5 and especially to Table 5.1 for an overview of the notation and data contained by an SRRP instance. In our learning-based destroy operator, the model π_θ outputs a value μ_{nd} for each $(n, d) \in N \times D$, indicating the probability of an employee-day pair (n, d) to be contained in the destroy set. More specifically, π_θ consists of two independently trained neural networks: one handling states containing infeasible solutions and one dealing with states containing feasible solutions. We decided to make this distinction as we observed that the behavior to learn can be quite different for feasible and infeasible solutions. Also, more training data is created for feasible solutions since the LNS spends more time in the feasible space. By considering two neural networks, we thus also avoid problems arising from the imbalance in training data. The only difference between the models in π_θ is the data used to train them. Hence, to improve readability, we will only refer to π_θ indicating the respective neural network throughout this whole section.

Another important aspect of our learning-based destroy operator is the temperature τ which is a strategy parameter regulating the influence of π_θ's output in the destroy set sampling process. To optimize the choice for τ, we introduce another model π^{T}_ϕ, which receives a state s^t and the output of π_θ as inputs and predicts a temperature τ for the current situation. Again, π^{T}_ϕ consists of two independently trained neural networks, one for infeasible and one for feasible solutions, and for the sake of readability, we will only refer to π^{T}_ϕ.

We explain the data generation process for each learning task in Section 7.6. Moreover, we define all the neural network architectures in Section 7.4 and review the relevant training procedures in Section 7.5. The next step in our learning-based destroy operator is the destroy set sampling process. Here, we use π_θ's outputs and the predicted temperature τ to select the employee-day pairs to be unassigned in the current solution. In Section 7.3, we provide the details regarding our destroy set sampling strategy. Eventually, in Section 7.7, we discuss the features used and state how they are extracted from a current state.

Our approach is inspired by Nair et al. [NBG+20] and Sonnerat et al. [SWK+21]. We will point out differences and similarities in the respective subsections. To summarize some of the key differences to those approaches: we propose a custom graph structure representing a current state, which allows us to apply a GNN even on this highly constrained COP. Moreover, we use an extra iteration in our data generation process for collecting data on more relevant states and introduce additional neural networks to learn optimal temperatures. Finally, we apply those temperatures in our destroy set sampling strategy tailored to the SRRP.

7.1 Markov Decision Process Formulation

As we have already stated in Section 4.4.1, an MDP is represented by a 4-tuple consisting of the set of states, the set of actions, a transition function, and a reward function. Let ST be the set of states and A the set of actions. In our setting, we define a state $s^t \in ST$ at step t of an episode to consist of an SRRP instance I and its current solution x^t_{nds}, where $n \in N, d \in D, s \in S$. An action $a^t \in \{0,1\}^{|N \times D|}$ at step t represents the selection of employee-day pairs to be destroyed. A positive assignment $a^t_{nd} = 1$ for an employee $n \in N$ and a day $d \in D$ indicates that employee day pair (n, d) is contained in the destroy set. Therefore, for such a tuple (n, d) and all $s \in S$, variables x^t_{nds} are unassigned in the current solution. Given a previous state s^t and an action a^t, the transition function $T : ST \times A \to ST$ determines the next state $T(s^t, a^t) = s^{t+1}$. Here, $T(s^t, a^t)$ is reached by destroying all variables associated with a_t in solution x^t_{nds} and then repairing the partial solution with the repair operator proposed in Section 6.2. We do not define a reward function since it does not play a role in our considerations.

7.2 Destroy Set Prediction as Conditional Generative Modeling

Before establishing our conditional generative modeling setting, we briefly discuss why we chose this approach over RL. The main issue with RL in our context is that it is vastly expensive to obtain enough training data since a single episode is equivalent to an entire LNS run. As the SRRP is a hard problem requiring longer computation times to reach high-quality solutions, the time needed to finish an episode also increases. Since today's RL algorithms typically require many episodes to develop their true strengths, we consider alternative strategies, such as our approach, to be more promising.

Inspired by Nair et al. [NBG+20], we propose a conditional generative model representing the distribution of actions (i.e., destroy sets) in a current state. For an arbitrary step t in an LNS run, consider a state $s^t \in ST$, an action $a^t \in A$, and the previously defined transition function T. Moreover, let $c : ST \to \mathbb{R}$ represent a function returning the objective value of a current solution x^t_{nds} stored in a state $s^t \in ST$ with respect to the SRRP objective function stated in (6.1). We define the following energy function over an action a^t:

$$E(a^t; s^t) = \begin{cases} c(T(s^t, a^t)) & \text{if } c(T(s^t, a^t)) < c(s^t), \\ \infty & \text{otherwise,} \end{cases} \tag{7.1}$$

which as in Nair et al. [NBG+20] defines the conditional distribution:

$$\pi(a^t | s^t) = \frac{\exp(-E(a^t; s^t))}{Z(s^t)}, \tag{7.2}$$

where $Z(s^t) = \sum_{(a')^t} \exp(-E((a')^t; s^t))$. Our learning efforts aim to approximate the conditional distribution in (7.2) utilizing a model $\pi_\theta(a^t | s^t)$ parameterized by θ. By using

the unscaled energy function as presented in (7.1), destroy sets leading to solutions with better (lower) objective values after repairing have a higher probability. And destroy sets not leading to improvements in objective value have zero probability. However, since our goal is to generate the best action in each state, we re-scale the energy function such that we assign probability $\pi((a^*)^t|s^t) = 1$ to an optimal action $(a^*)^t$ and a probability of zero to each other action in state s^t. Sonnerat et al. [SWK+21] adapted the conditional generative modeling design from Nair et al. [NBG+20] in the same way. As a consequence, we only have to consider training data containing optimal actions. We describe how we obtain such optimal or near-optimal labels in Section 7.6.

7.3 Sampling Destroy Sets

In Section 6.3, we introduced the randomized destroy operator, which selects z_1 employee-day pairs $(n, d) \in N \times D$ and destroys all the variables associated with employee-day pairs within the range of z_2 days before to z_2 days after d. Remember that $z_1 \in \mathbb{N}_0$ and $z_2 \in \mathbb{N}_0$ are strategy parameters. Moreover, we described that selecting employee-day pairs without this range does not give good results since the SRRP consists of constraints regarding consecutive working assignments. These constraints are HCWSA ensuring that an employee is not assigned to less or more than a specified number of consecutive working assignments, and HCSTA enforcing that for each employee, the number of consecutive assignments to a specific shift is within an allowed range. Hence, we need to select consecutive days to improve a solution, which is not probable enough if randomly selecting single employee-day pairs at a time. Although in our learning-based destroy operator, the neural network π_θ outputs a value for each employee-day pair describing its probability to be in the destroy set, we encounter the same issues. Therefore, we cannot directly apply the approach of Sonnerat et al. [SWK+21], which would construct a destroy set $\mathcal{U}$ by iteratively adding an employee day pair (n, d) with probability proportional to

$$(\pi_\theta(a^t_{nd} = 1 \mid s^t) + \epsilon)^{\frac{1}{\tau}} \cdot \mathbb{I}[(n, d) \in \mathcal{U}], \tag{7.3}$$

where τ is a temperature parameter to strengthen or weaken the influence of the neural network and $\epsilon > 0$ is an offset to give every pair a non-zero probability. This approach ensures that the size of the destroy set can be predetermined. Note that Sonnerat et al. [SWK+21] refrain from sampling destroy decisions directly from the Bernoulli distributions since this often leads to small destroy sets. We also independently observed this behavior in our experiments. It results from the unbalanced data as only a small number of employee-day pairs is part of the destroy set. Hence, the neural network predicts a lower probability for each pair.

To counter the discussed challenges, we propose a new destroy set sampling strategy, where we select a block of consecutive employee-day pairs. We assign each pair $(n, d) \in N \times D$ a weight

$$w_{nd} = \sum_{d'=\max(1,\, d-z_2)}^{\min(|D|,\, d+z_2)} (\pi_\theta(a^t_{nd'} = 1 \mid s^t) + \epsilon)^{\frac{1}{\tau}} \cdot \mathbb{I}[(n, d') \in \mathcal{U}], \tag{7.4}$$

which can be interpreted as the sum of all neural network outputs for employee n in a window of $2z_2 + 1$ consecutive days around day d. Then, we randomly select an employee-day pair (n, d) proportional to its weight w_{nd} and add pairs (n, d') for all $d' \in \{\max(1, d - z_2), \ldots, d, \ldots, \min(|D|, d + z_2)\}$ to $\mathcal{U}$. We repeat this process z_1 times. Eventually, for each $(n, d) \in \mathcal{U}$, we unassign variables $x^t_{nd's}$ for each shift $s \in S$ in the current solution.

7.4 Neural Networks

Applying classical neural network models such as the MLP, CNN, or RNN has the disadvantage that the order of the input features impacts the output of the model. Moreover, these models, except the RNN, only work on inputs of equal dimensions. GNNs do not impose such restrictions and even make it possible to model the structure of a problem as a graph. For these reasons, we also rely on a GNN as our ML model of choice. A GNN parameterizes a graph enabling it to learn from data on the graph. However, the SRRP is not directly defined on a graph. Addanki et al. [ANA20] propose a learning LNS framework for solving MILPs in general. They use a CVIG [AGCL12] which is a bipartite graph where one partition contains a node for each variable and the other a node for each constraint in the MILP. An edge exists between a variable and a constraint node if the variable occurs in the constraint. A corresponding CVIG for practical instances of the SRRP would be huge. In preliminary tests, we observed that this huge graph makes it inefficient to compute outputs and update the network parameters.

7.4.1 Custom Graph Representation

We propose the following custom graph representation for the SRRP. This graph representation is not a reformulation of the SRRP as a graph problem, but it reflects a knowledge graph containing information for choosing promising employee-day pairs. We define this graph as a structure $G = (V, E, X)$, where V is the set of nodes, E the set of edges, and $X \in \mathbb{R}^{|V| \times p}$ a node feature matrix which assigns each node $v \in V$ a p-dimensional feature vector $X_v = (X_{v,1} \ldots X_{v,p})$. The set of nodes $V = V_{\text{emp}} \cup V_{\text{assign}} \cup V_{\text{day}}$ is composed of three different types which are employee $V_{\text{emp}} = \{n \mid n \in N\}$, assignment $V_{\text{assign}} = \{(n, d) \mid n \in N, d \in D\}$, and day $V_{\text{day}} = \{d \mid d \in D\}$ nodes. The assignment nodes represent the employee-day pairs. There are edges between an assignment node and its associated employee and day nodes. Moreover, since the days represent a sequence, we add edges between each day and the day that directly follows it. Hence, $E = \{(v, u) \mid v \in V_{\text{emp}}, u = (n, d) \in V_{\text{assign}}, v = n\} \cup \{(v, u) \mid v \in V_{\text{day}}, u = (n, d) \in V_{\text{assign}}, v = d\} \cup \{(v, u) \mid v, u \in V_{\text{day}}, u = v + 1\}$. Finally, we provide the actual features X extracted from a state s^t in Section 7.7. In Figure 7.2, we show an example graph of an SRRP instance with three employees and days to provide more intuition for this graph structure. The interpretation of this representation is that an employee is involved in an assignment and this assignment takes place on a specific day in the planning horizon. A

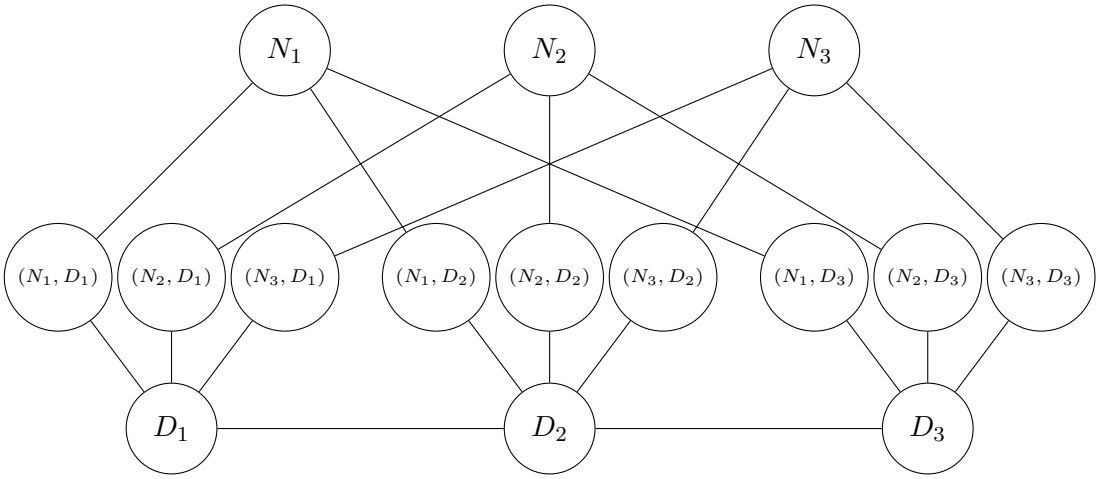


Figure 7.2: Graph-based representation of the SRRP on an example instance with three employees and three days. There is a node for each employee (N_1, N_2, N_3) and each day (D_1, D_2, D_3). For each employee-day pair (N_i, D_j), there is a node, which is connected to the employee node N_i and the day node D_j. Each day D_i is connected to the day that follows it $D_{(i+1)}$, if such a day exists in the planning horizon.

state $s^t \in ST$, consisting of an SRRP instance and its current solution x^t_{nds}, holds all the required information to create such a graph structure. If we later use a state directly as an input to our neural network, we mean that it has first been transformed into such a graph representation.

Classical GNNs such as the GCN [KW17], GIN [XHLJ19], or graph attention network (GAT) [VCC+18] assume that the given graph is homogeneous, meaning that all nodes are of the same type and therefore contain the same features. Our proposed graph structure, however, comprises three different types of nodes. The main reason for this design choice is the size of the graph since it directly influences the run times of the GNN forward pass. With the suggested graph structure, there are $O(|N| \cdot |D|)$ nodes and $O(|N| \cdot |D|)$ edges. For a homogeneous graph consisting of employee-day pair nodes, there would be at least $O(|N|^2 \cdot |D|)$ edges. The reason is that this graph requires edges between the employee-day pair nodes of the same day to share information concerning the cover requirements of that day. However, it is not sensible to leave out nodes of the same day in this connection. In total, this results in $|D|$ complete subgraphs having $|N|^2$ edges. In our proposed structure, the employee and day nodes can naturally hold features that aggregate information for an employee or a day. For example, employee nodes might contain the number of assignments to working shifts of an employee, and day nodes the number of employees assigned to a specific shift on a day as features.

One way to address that classical GNNs assume homogeneous graphs would be to apply the relational graph convolutional network (R-GCN) proposed by Schlichtkrull et al. [SKB+18]. The R-GCN works on relational data expressible as triples of the form (subject,

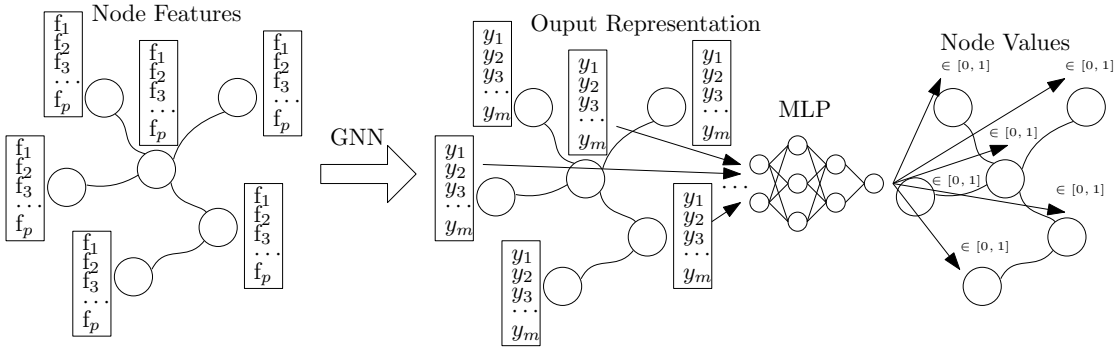


Figure 7.3: Simplified representation of the applied neural network architecture for the primary model π_θ. The figure is inspired by [CCCR21].

predicate, object). Then, we could transform our graph by directing and labeling the edges to mimic relational data, e.g. (employee, assignedTo, assignment) and (assignment, takesPlaceOn, day). However, this approach would require a custom implementation and prevent us from using existing optimized GNN libraries. Therefore, we decided to follow another strategy, which is to use the same feature vector for all nodes, consisting of the union of all node-type-specific features plus the node type encoded with a one-max encoding. Let $\mathrm{f}_1^{\mathrm{emp}}, \ldots, \mathrm{f}_{q^{\mathrm{emp}}}^{\mathrm{emp}}$ be the employee features, $\mathrm{f}_1^{\mathrm{assign}}, \ldots, \mathrm{f}_{q^{\mathrm{assign}}}^{\mathrm{assign}}$ the assignment features, and $\mathrm{f}_1^{\mathrm{day}}, \ldots, \mathrm{f}_{q^{\mathrm{day}}}^{\mathrm{day}}$ the day node features, where $q^{\mathrm{emp}}, q^{\mathrm{assign}}, q^{\mathrm{day}}$ are the number of employee, assignment, and day node features, respectively. Then each feature vector x_v of a node v, independent of the node type, is of the form

$$(\mathrm{f}_1^{\mathrm{emp}}, \ldots, \mathrm{f}_{q^{\mathrm{emp}}}^{\mathrm{emp}}, \mathrm{f}_1^{\mathrm{assign}}, \ldots, \mathrm{f}_{q^{\mathrm{assign}}}^{\mathrm{assign}}, \mathrm{f}_1^{\mathrm{day}}, \ldots, \mathrm{f}_{q^{\mathrm{day}}}^{\mathrm{day}}, \mathrm{f}_1^{\mathrm{enc}}, \mathrm{f}_2^{\mathrm{enc}}, \mathrm{f}_3^{\mathrm{enc}}) \tag{7.5}$$

where $\mathrm{f}_1^{\mathrm{enc}}, \mathrm{f}_2^{\mathrm{enc}}, \mathrm{f}_3^{\mathrm{enc}} \in \{0, 1\}$ is the mentioned one-max encoding indicating whether the node is an employee, assignment, or day node, respectively. For example, if a node is an employee node, its feature vector contains the employee's values for $\mathrm{f}_1^{\mathrm{emp}}, \ldots, \mathrm{f}_{q^{\mathrm{emp}}}^{\mathrm{emp}}$, zeros for the assignment and day features, and the associated one-max encoding. In Section 7.7, we provide and discuss the features used.

7.4.2 Architectures

So far, we have established the underlying graph structure. Now, we introduce the GNN used to parameterize this graph and the network architecture for the destroy set model π_θ utilized to obtain the weights for selecting employee-day pairs. Figure 7.3 shows a simplified representation of this neural network architecture. First, we employ a basic GNN similar to the neural network for graphs (NN4G) [Mic09] architecture. The NN4G is the first work towards spatial-based ConvGNNs [WPC+20]. Our GNN updates the feature representation $H^{(l)}$ in layer l by applying the update function

$$H^{(l)} = \sigma\left(H^{(l-1)}W_1^{(l)} + AH^{(l-1)}W_2^{(l)} + b^{(l)}\right), \tag{7.6}$$

where $H^{(0)} = X$, A is the adjacency matrix, σ is a non-linear activation function, $b^{(l)} \in \mathbb{R}^q$ is the learnable bias, and $W_1^{(l)}, W_2^{(l)} \in \mathbb{R}^{m \times q}$ denote the weight matrices for layer l, where $m, q \in \mathbb{N}_0$ are the input and output feature dimensions, respectively. Applying the GNN to an input yields the final node representation $H^{(L)}$, where $H_v^{(L)}$ is the final vector representation for each node $v \in V$. Our neural network architecture additionally consists of an MLP, which utilizes a sigmoid activation in the last layer. This MLP is applied to each final vector representation $H_v^{(L)}$ for all $v \in V$. As a result, we receive a single value between 0 and 1 for each node. Note that in Figure 7.3, the final vector representations are depicted by y. We only require the final output for the assignment nodes. Hence, it would be sufficient to employ the MLP on their final vector representations only. We still applied it on all the nodes since in our thesis, only about 4% of the nodes are not assignment nodes, and since we have an efficient implementation being faster without filtering out the other nodes. Nevertheless, it might be different for other works.

To make the connection to our conditional generative modeling approach and the conditional distribution π from Section 7.2, let π_θ be our previously presented neural network, where θ are all the learnable parameters, including the GNN and MLP weights. Remember that an action a_{nd}^t at step t indicates whether an employee-day pair $(n, d) \in N \times D$ is contained in the destroy set ($a_{nd}^t = 1$) or not ($a_{nd}^t = 0$). Also, remember that the sets V_{assign} and $N \times D$ are isomorphic, meaning that there is a one-to-one correspondence between each assignment node and employee day pair $(n, d) = v \in V_{\text{assign}}$. As Nair et al. [NBG+20] and Sonnerat et al. [SWK+21], we define π_θ to be a conditional-independent model of the form

$$\pi_\theta(a^t \mid s^t) = \prod_{(n,d) \in V_{\text{assign}}} \pi_\theta(a_{nd}^t \mid s^t), \tag{7.7}$$

which, given a state s^t, predicts the probability of an employee-day pair (n, d) being contained in the destroy set independently of the other employee-day pairs. The probability $\pi_\theta(a_{nd}^t \mid s^t)$ is a Bernoulli distribution, and we compute its success probability μ_{nd} as

$$t_{nd} = \texttt{MLP}(H_{(n,d)}^{(L)}; \theta), \tag{7.8}$$

$$\mu_{nd} = \pi_\theta(a_{nd}^t = 1 \mid s^t) = \frac{1}{1 + \exp(-t_{nd})}. \tag{7.9}$$

Despite Nair et al. [NBG+20] pointed out in their work that it is not possible to accurately model a multi-modal distribution using the assumption of conditional independence, they could still report strong empirical results. Mathematically more accurate alternatives are autoregressive models [Ben00] or inferring one employee-day pair at a time by repeatedly evaluating the neural network. However, this increased accuracy comes at the cost of significantly slower inference times [NBG+20, SWK+21], which are not reasonable in our setting.

The architecture of the temperature neural network π_ϕ^{T} is similar to the previously discussed architecture of the destroy set model π_θ. However, there are some key differences,

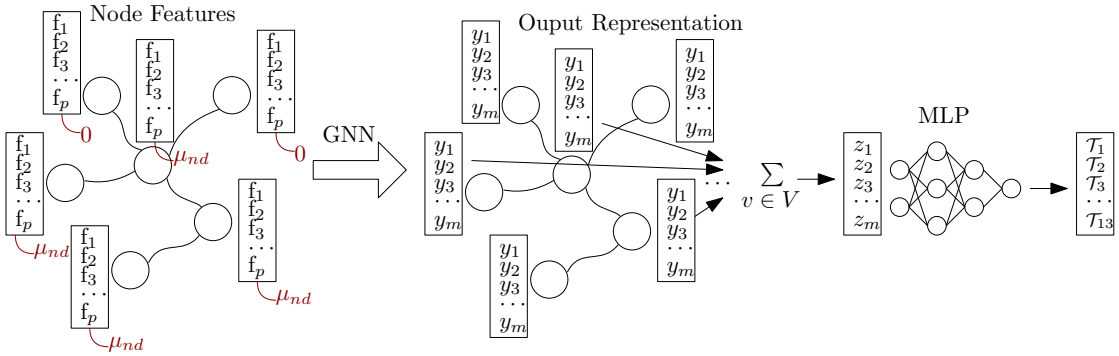


Figure 7.4: Simplified representation of the temperature model π_ϕ^{T} neural network architecture. We add the outputs of the destroy set model π_θ to the node features. The final MLP returns a vector containing a probability for each temperature $\tau \in \mathcal{T}$ to be the best selection.

which can already be seen in the simplified representation shown in Figure 7.4. The temperature model π_ϕ^{T} evaluates if the influence of π_θ on the destroy set sampling process should increase or decrease in a specific state s^t. Therefore, we add the output of π_θ as an additional feature to the node features of the graph representation. More specifically, we append $\mu_{nd} = \pi_\theta(a_{nd}^t = 1 \mid s^t)$ for assignment nodes $(n, d) = v \in V_{\text{assign}}$ and zero for every other node $v \in V_{\text{emp}} \cup V_{\text{day}}$. Another difference is that we employ a read-out layer on the final node representations $H_v^{\mathrm{T},(L)}$, which summarizes the information into one vector by applying $\sum_{v \in V} H_v^{\mathrm{T},(L)}$. Here, we use $H_v^{\mathrm{T},(L)}$ to represent the final node representations of the temperature GNN for each $v \in V$. In Figure 7.4, these are depicted by y again for simplicity. Finally, we utilize an MLP with a softmax function in the last layer to return a probability for each temperature $\tau \in \mathcal{T}$ to be the best selection.

7.5 Training

The training set for the destroy set model π_θ is represented by $\mathcal{D}_{\text{train}} = \{(s_{(j)}^{1:T_j}, a_{(j)}^{1:T_j})\}_{j=1}^{M}$ containing the data from M sampled trajectories. For each such trajectory $j \in \{1, \dots, M\}$ consisting of T_j steps, $\{s_{(j)}^t\}_{t=1}^{T_j}$ are the states and $\{a_{(j)}^t\}_{t=1}^{T_j}$ the corresponding expert actions or destroy sets. We learn the weights θ of our model π_θ by minimizing the loss function

$$\mathcal{L}(\theta) = -\sum_{j=1}^{M} \sum_{t=1}^{T_j} \log \pi_\theta(a_{(j)}^t \mid s_{(j)}^t), \tag{7.10}$$

which is the negative log likelihood of the expert actions.

The training set $\mathcal{D}_{\text{train}}^{\mathrm{T}} = \{(s_{(j)}^{1:T_j}, o_{(j)}^{1:T_j}, y_{(j)}^{1:T_j})\}_{j=1}^{M^{\mathrm{T}}}$ for the temperature model π_ϕ^{T} contains the outputs $\{o_{(j)}^t\}_{t=1}^{T_j}$ of π_θ in the respective states $\{s_{(j)}^t\}_{t=1}^{T_j}$ for each sampled trajectory $j \in \{1, \dots, M^{\mathrm{T}}\}$. The associated labels $\{y_{(j)}^t\}_{t=1}^{T_j}$ consist of a one-max encoding of the best

temperature for each time step t of a trajectory j. Eventually, we optimize the weights ϕ by minimizing the cross-entropy loss

$$\mathcal{L}^{\mathrm{T}}(\phi) = -\sum_{j=1}^{M^{\mathrm{T}}} \sum_{t=1}^{T_j} y_{(j)}^t \log \pi_\phi^{\mathrm{T}}(s_{(j)}^t, o_{(j)}^t). \tag{7.11}$$

We perform each training in mini-batches of size 32. In addition to $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{train}}^{\mathrm{T}}$, we also create validation sets of the same form as the respective training sets containing about a fourth of the total generated trajectories. We hold out this data from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{train}}^{\mathrm{T}}$ to evaluate the progress on unseen data during training. Furthermore, we use early stopping [GBCB16, p. 246], which Geoffrey Hinton called a "beautiful free lunch" [Gér19, p. 141], to avoid overfitting. Here, the training is stopped as soon as the loss or another metric of choice stays above an observed minimum for a certain number of training iterations. Then, the model achieving the best score is selected as the final model. As our optimizer, we use ADAM [KB17] with a learning rate of 0.001 and an exponential decay rate of 0.9 for the first and 0.999 for the second momentum.

7.6 Training Data Generation

Our data generation process is inspired by the expert policy from Sonnerat et al. [SWK+21], which uses local branching [FL03] to create optimal destroy sets in a given state. In local branching, a constraint is added to the MILP that allows at most η decision variables to change compared to a given previous solution. In the course of this section, we refer to the MILP extended with the local branching constraints as extended MILP or local branching-based MILP. If this extended MILP is solved to optimality in a current state s^t, an optimal destroy set can be generated by comparing the old solution x_{nds}^t with the new solution $x_{nds}^{(t+1)}$ for all $n \in N, d \in D, s \in S$ and collecting the variables that changed. Since our destroy sets do not directly consist of decision variables but employee-day pairs, we define the following additional constraints

$$\sum_{(n,d,s) \in N \times D \times S:\, x_{nds}^t = 0} x_{nds}^{(t+1)} + \sum_{(n,d,s) \in N \times D \times S:\, x_{nds}^t = 1} (1 - x_{nds}^{(t+1)}) \leq 2\eta, \tag{7.12}$$

which ensure that at most η employee-day pairs change compared to the previous solution. Given an employee-day pair (n, d) for an employee $n \in N$ and a day $d \in D$, the set $\{x_{nds} \mid \forall s \in S\}_{nd}$ contains all the decision variables represented by this pair. A modification of a variable x_{nds} leads to two changes in total since the hard constraint HOSPD enforces that each employee must be assigned to exactly one shift, including the free shift, on each day. Hence, we need to multiply η by two on the right-hand side of (7.12). In the following, we refer to solving the local branching-based MILP and applying the best extracted destroy set as our expert policy.

To generate training samples for the destroy set model π_θ, we modify and extend the behavior cloning [Pom88] procedure of Sonnerat et al. [SWK+21]. In behavior cloning,

Algorithm 8: DAGGER

$\mathcal{D} \leftarrow \emptyset$
$\hat{\pi}_1 \leftarrow$ expert policy π^*
foreach $i = 1, 2$ **do**
  sample trajectories for randomly sampled SRRP instances using $\hat{\pi}_i$ in parallel
  create dataset $\mathcal{D}_i = \{(s^t, \pi^*(s^t))\}$ containing all states s^t visited in the sampled trajectories and expert actions $\pi^*(s^t)$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
  train model π_θ on $\mathcal{D}$
  $\hat{\pi}_{i+1} \leftarrow \pi_\theta$
end
return $\hat{\pi}_3$

an expert policy is used to sample trajectories for training instances. Then, a model is trained on the expert actions in the encountered states to mimic the expert policy. Sampling a trajectory using a policy $\hat{\pi}$ means that in each state s^t at step t of an episode (LNS run), we store state s^t, use $\hat{\pi}$ to create a destroy set $a^t = \hat{\pi}(s^t)$, and move to the next state $s^{t+1} = T(s^t, a^t)$. For a learned policy π_θ, we apply the destroy set sampling strategy from Section 7.3 with temperature $\tau = 1$ to generate a destroy set in a current state.

The caveat of behavior cloning is that the visited trajectory states solely depend on the expert policy. Since it is hard to train a model completely mimicking the expert policy in every situation, we probably encounter unfamiliar states when following the trained policy. In preliminary experiments, we encountered issues arising from this problem, as our models sometimes got stuck in challenging states for which the expert policy did not collect enough reference data. To avoid such problems, we propose two strategies:

(a) The first one is to extend the behavior cloning algorithm for the destroy set model by an additional iteration. In this iteration, we use the trained behavior cloning model to sample new trajectories and save the encountered states together with the expert policy actions as new training data. Then, we train a new model on the data aggregated over all iterations. Algorithm 8 shows the pseudocode of this procedure, where the first iteration is equivalent to behavior cloning. This extension of behavior cloning is called Dataset Aggregation (DAGGER) and was proposed by Ross et al. [RGB11]. Usually, one applies more than two iterations of DAGGER. However, computing the expert policy action is expensive since we have to solve the extended MILP. Therefore, we decided to take measures to make the data generation more efficient. First, we terminate solving the local branching-based MILP in a state after a time limit. Second, we execute the trajectory sampling for each SRRP training instance in parallel. And lastly, we only create a training sample for every third visited state when sampling trajectories with a trained model if the solution is feasible. The SRRP training instances are generated on the fly by randomly sampling new disruptions as described in Section 8.1.

In the results presented in Section 8, we will show that the enhanced data generation algorithm slightly improves the performance for the LNS with the learned destroy operator. However, there are still some states in the test instances for which the trained models struggle to find good destroy sets. Possibly, additional DAGGER iterations could solve this issue. However, we propose another, more efficient strategy.

(b) This strategy involves training an additional model π_ϕ^{T} with parameters ϕ which optimizes the temperature parameter τ for the destroy set sampling process. The temperature τ determines the influence of the primary model π_θ on selecting employee-day pairs for the destroy set. While a low temperature, $\tau < 1$, increases the impact of π_θ, a higher one, $\tau > 1$, decreases it. We refer to Section 7.3 for a detailed description of the destroy set sampling process. Therefore, π_ϕ^{T} acts as an evaluator of the main model. If it recognizes that the outputs of π_θ are bad in a specific state s^t, it predicts a higher value for τ to diminish the influence of π_θ and vice versa. We treat the learning task as a classification problem and provide the details about the training of π_ϕ^{T} and π_θ in Section 7.5. For the data generation, we predefine a set of temperatures

$$\mathcal{T} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 5\}. \tag{7.13}$$

Next, we produce trajectories using π_θ, where we apply each temperature $\tau \in \mathcal{T}$ to sample three destroy sets in a state s^t at step t of an episode. Then, we save s^t, $\pi_\theta(s^t)$, and the temperature τ^* creating the best destroy sets on average. To evaluate the best average performance, we consider the objective value of the newly created solution if the solution has been improved. Otherwise, we consider an upper bound value, the objective value of the initially constructed solution, in the evaluation. Therefore, we prefer temperature values consistently improving the solution over temperatures that improve the solution well once but have a lower probability of improving it in general. Finally, we move to the next state $T(s^t, a^t_{\tau^*})$, using an action sampled with τ^*. Algorithm 9 shows the outlined temperature data generation process for one SRRP instance. Note that a current state consists of an SRRP instance and its current solution as defined in Section 7.1. In practice, we perform multiple executions of Algorithm 9 in parallel to obtain all the temperature training data.

7.7 Node Features

For all the different node types our graph structure G, which are employee V_{emp}, assignment V_{assign}, and day V_{day} nodes, we assemble a set of SRRP-specific features summarized in Table 7.1. Most of these features are dynamic and refer to the current solution x^t_{nds} stored in a state s^t.

We state and describe the features in the order they occur in the feature vector and start by introducing the employee node features. These include the total number of working assignments of an employee, the total number of working assignments of an employee minus the minimum allowed number of working days ($\alpha_{\min}$), and the maximum allowed number of working days ($\alpha_{\max}$) minus the total number of working assignments of an

Algorithm 9: Temperature Data Generation

Input: fully trained destroy set model π_θ

$\mathcal{D} \leftarrow \emptyset$
$I \leftarrow$ randomly sample SRRP instance
$x_{nds} \leftarrow$ `ConstructSolution`(I)
$ub \leftarrow c(x_{nds})$ // $c : \mathcal{S} \to \mathbb{R}$ objective function
while *true* **do**
  $S_\tau \leftarrow 0$ for every $\tau \in \mathcal{T}$
  $X_\tau \leftarrow \emptyset$ for every $\tau \in \mathcal{T}$
  foreach $\tau \in \mathcal{T}$ **do**
    foreach $i = 1, 2, 3$ **do**
      $x'_{nds} \leftarrow$ copy current solution x_{nds}
      $a_\tau \leftarrow$ sample action with $\pi_\theta((I, x'_{nds}))$ and τ (see Section 7.3)
      $x'_{nds} \leftarrow$ extract solution from $T((I, x'_{nds}), a_\tau)$
      if $c(x'_{nds}) < c(x_{nds})$ **then**
        add $c(x'_{nds})$ to S_τ
        add x'_{nds} to X_τ
      else
        add ub to S_τ
      end
    end
  end
  break while-loop **if** `IsEmpty`(X_τ) for all $\tau \in \mathcal{T}$
  $\tau^* \leftarrow \text{argmin}_{\tau \in \mathcal{T}} S_\tau$
  save $((I, x_{nds}), \pi_\theta((I, x_{nds})), \tau^*)$ in $\mathcal{D}$
  $x_{nds} \leftarrow$ randomly sample action from X_{τ^*}
end
return $\mathcal{D}$

employee. Moreover, they contain the total number of assignments to each shift $s \in S$ of an employee, the total number of assignments to each shift $s \in S$ of an employee minus the minimum allowed number of assignments to this shift s ($\gamma_s^{\min}$), and the maximum allowed number of assignments to shift $s \in S$ ($\gamma_s^{\max}$) minus the total number of assignments to this shift s of an employee. The final information stored for an employee node is the total number of whole day absences and the total number of absences for each shift $s \in S$ of an employee. These features should provide the ML model with the required information regarding the workload constraints (HWSA and HSTA) of the SRRP. To normalize and hence bring the whole input data into similar scales, we divide all the employee node features by the number of days $|D|$.

Next, we describe the features of the assignment nodes, where each node represents an employee-day pair (n, d) for an $n \in N$ and $d \in D$. There, the first $|S|$ features are

Table 7.1: Description of the features. The Count column specifies how many features derive from a description.

Feature Description	Count
Employee n	
- total number of working assignments of employee n	1
- total number of working assignments of employee n minus minimum number of working days in the planning horizon ($\alpha_{\min}$)	1
- maximum number of working days in the planning horizon ($\alpha_{\max}$) minus total number of working assignments of employee n	1
- total number of assignments to shift $s \in S$ of employee n	$\|S\|$
- total number of assignments to shift $s \in S$ of employee n minus minimum allowed number of assignments to this shift s ($\gamma_s^{\min}$)	$\|S\|$
- maximum allowed number of assignments to shift $s \in S$ ($\gamma_s^{\max}$) minus total number of assignments to this shift s of employee n	$\|S\|$
- total number of whole day absences of employee n	1
- total number of absences per shift $s \in S$ of employee n	$\|S\|$
Assignment (n, d)	
- flag indicating whether employee n is assigned to shift $s \in S$ on day d	$\|S\|$
- flag indicating whether employee n is assigned to shift $s \in S$ on day d in the original roster	$\|S\|$
- flag indicating whether employee n is absent on shift $s \in S$ on day d	$\|S\|$
- flag indicating whether the minimum number of consecutive working days constraint is violated for employee n on day d	1
- flag indicating whether the maximum number of consecutive working days constraint is violated for employee n on day d	1
- flag indicating whether the minimum number of consecutive assignment constraint is violated for employee n on day d and shift $s \in S$	$\|S\|$
- flag indicating whether the maximum number of consecutive assignment constraint is violated for employee n on day d and shift $s \in S$	$\|S\|$
Day d	
- total number of assignments to each shift $s \in S$ on day d	$\|S\|$
- total number of assignments to each shift $s \in S$ on day d minus cover requirements for this shift s on day d (R_{ds}^{c})	$\|S\|$
Type Encoding	
- flag indicating whether the node is an employee, assignment, or day node (onemax encoded)	3

a one-max encoding indicating to which shift $s \in S$ employee n is assigned on day d. Similarly, the assignment node features include another one-max encoding stating the assigned shift $s \in S$ of employee n on day d in the original roster. Furthermore, they include an one-max encoding showing whether employee n is absent on shift $s \in S$ on day d. With these features, we want to supply the model with the information concerning the changes to the original schedule (SMOD) and the shifts an employee is absent (HABSE). More assignment node features are dealing with the workload constraints for consecutive days. These include flags indicating whether the minimum and the maximum number of consecutive working days constraints (HCWSA) are violated for employee n on day d. And two one-max encodings stating whether the minimum and the maximum number of consecutive working assignments to a shift $s \in S$ (HCSTA) are violated for employee n on day d.

Finally, the last node type is the day node for a day d. The day node features include the total number of assignments to each shift $s \in S$ on day d and the number of total assignments to each shift $s \in S$ on day d minus the cover requirements for this shift s on day d. To normalize the day node features, we divide them by the number of employees $|N|$. Hence, these features hold the information regarding under- and overstaffing of all the shifts per day. Lastly, the last three features are represented by a one-max encoding indicating whether the given node is an employee, assignment, or day node.

As we have already described in Section 7.4, each node, independent of its type, is equipped with a feature vector of the same form as shown in (7.5). This feature vector consists of the union of all the node-type features plus the one-max node type encoding. For example, if a node is an employee node, it contains its respective features for employee nodes, zeros for the other node type features, and the associated node type encoding. Note that for models only dealing with already feasible solutions, the assignment node features concerning constraint violations would all constantly have a value of zero. Hence, we remove them for those models and reduce the feature space size from $11 \cdot |S| + 9$ to $9 \cdot |S| + 7$.

CHAPTER 8

Computational Results

In this section, we present the computational results for the described optimization algorithms and the proposed destroy operators. First, we introduce the test instance generation process and state the parameters used for these instances in Section 8.1. Then, in Section 8.2, we establish the configurations used for the computational experiments and finally present the obtained results.

8.1 Test Instances

Benchmark or test instances are needed to examine the robustness and performance of algorithms. To the best of our knowledge, no SRRP benchmark instance dataset meeting our requirements is available. There are two main issues with the existing datasets from Moz and Pato [MP07] and Wickert et al. [WSB19]. First, their goal is slightly different from ours. While we seek to find fast and high-quality solutions for one specific schedule suffering from disruptions, they deal with the occurrence of disruptions to various schedules. Hence, their datasets do not contain any rosters for which there are sufficiently many disruption variants. Second, even if we selected one of their schedules and sampled new disruptions for it, those instances would not be hard enough. From the results of Wickert et al. [WSB19], one can conclude that the instances of both datasets can be solved to optimality rather quickly with a state-of-the-art MILP solver. This especially holds for instances with a planning horizon of four weeks, which is the setting we consider in our work. However, the benefits of the LNS meta-heuristics lie in the ability to find high-quality solutions for problems where exact methods are not efficient enough. Therefore, we require more challenging instances for our research.

Based on this argumentation, we decided to create our own benchmark instance generation process. We apply this approach to create 60 validation and 120 test instances for a randomly created base schedule with 110 employees. We do not specify a fixed number of training instances since we generate them on the fly by randomly sampling new

disruptions for this base schedule. Furthermore, we use the same generation process to create 30 test instances for randomly created schedules with 120, 130, 140, and 150 employees. We use these 120 instances to evaluate how our learning-based algorithm generalizes to different schedules. In the following explanations, we will refer to the instance generation regarding the base schedule with 110 employees. However, this approach works equivalently for instances with different numbers of employees.

Our instance generation approach is based on a three-step process:

1. Generate a hard staff rostering or NRP instance.
2. Solve the staff rostering instance by creating a high-quality schedule.
3. Introduce disruptions to the schedule.

Generating Hard Staff Rostering Instances For the first step, we implemented the nurse scheduling problem instance generation algorithm (NSPGen) from Vanhoucke and Maenhout [VM09]. This algorithm allows us to create randomized NRP instances with specific characteristics. These instances consist of a coverage requirements matrix and a preference matrix. The coverage requirements matrix specifies the number of employees needed for each shift on all days. In the preference matrix, all employees have a value between 1 and $|S|$ (number shifts) for each shift on each day, indicating their desire to do this shift. The lower the number, the more desirable it is for an employee to work a shift. The characteristics of these matrices are described by indicators belonging to three classes: problem size, preference distribution measures, and coverage distribution measures. The problem size class contains the number of employees $|N|$, days $|D|$, and shifts $|S|$ as indicators. The preference distribution measures are described by the NPD, SPD, and DPD indicators. Finally, the TCC, DCD, and SCD are the coverage distribution measures. Each of these six distribution indicators can take a value between 0 and 1 to determine the characteristics of an instance. In Table 8.1, we summarize the indicators and state their interpretations. For more details on the formulas behind them, we refer to Vanhoucke and Maenhout [VM09]. Some indicators have conflicting goals. Therefore, it is not possible to realize all indicator combinations. This should be taken into consideration if a high accuracy of the indicators is required. The right-most column in Table 8.1 shows the selected complexity indicator values, which we used to create our NRP instance. We determined these values based on experiments with the goal of creating a hard and, at the same time, realistic class of instances. Regarding the indicators of the problem size class, we chose $|N| = 110$, $|D| = 28$, and $|S| = 4$ for the number of employees, days, and shifts, respectively. Furthermore, we set the preference distribution measures NPD, SPD, and DPD to 0.5. Lastly, for the coverage distribution, we used TCC 0.85, DCD 0.15, and SCD 0.2 to create our NRP instance.

Solving the Staff Rostering Instances Once the NRP instance is generated, the next step is to construct a feasible and high-quality schedule for it. For this purpose, we

Table 8.1: Summary of the NSPGen algorithm [VM09] indicators including their interpretations. The Selected Values column shows the considered complexity indicators in the NRP instance generation process. Adapted from Vanhoucke and Maenhout [VM09].

Indicator	**Interpretation**	**Selected Values**
$\|N\|$	number of employees	110
$\|D\|$	number of days in the planning horizon	28
$\|S\|$	number of shifts including the free shift	4
NPD	nurse preference distribution (NPD) = 0: there is no clear preference for a particular shift among all employees NPD = 1: there is a clear preference among all employees for a particular shift	0.5
SPD	shift-preference distribution (SPD) = 0: all employees express indifference between the shifts SPD = 1: each employee expresses a clear preference ranking among the individual shifts	0.5
DPD	day preference distribution (DPD) = 0: all employees express a similar preference or aversion for similar shifts over all days DPD = 1: each employees has assigned a different preference value for similar shifts over the days	0.5
TCC	total coverage constrainedness (TCC) = 0: the amount of required employees is 0 for all shifts TCC = 1: the average total daily coverage is equal to the number of employees	0.85
DCD	day coverage distribution (DCD) = 0: the coverage requirements are equally distributed among all days DCD = 1: the coverage requirements are maximal for one or several days (depending on TCC), and zero for all remaining	0.15
SCD	shift coverage distribution (SCD) = 0: the coverage requirements are equally distributed among all shifts of a single day SCD = 1: The coverage requirements are maximal (depending of DCD) for a single shift, while all other shifts on that day do not need employees	0.2

have the MILP from Section 5.2.1. We solve this MILPs using a time limit of two hours to obtain the schedules.

Introducing Disruptions to Schedules The final step consists of introducing disruptions to the constructed schedules. In Section 2, we defined three types of disruptions: single-shift, multi-shift, and demand disruptions. According to the Austrian Bureau of Statistics, the average length of sick leave was 9.7 days in Austria in 2019 [Sta]. Since illness and vacation are the main categories of multi-shift disruptions and ten also sounds reasonable for the average length of vacations, we model them in the same step. Therefore, we use the binomial distribution $B(28, 0.35)$, which has a mean of 9.8, to represent the distribution of the length of sick leaves and vacations. Figure 8.1 shows the probability mass function of $B(28, 0.35)$. To incorporate the disruptions into the schedules, we select a random employee $n \in N$ and a random day $d \in D$. Then, we draw a random number L from $B(28, 0.35)$, and for all $l \in \{0, \dots, L-1\}$ set $U^{\text{msd}}_{n(d+l)} = 1$ to indicate that employee n is absent on day $d + l$. We repeat this process until there are $|N|$ days of absences in total, which means that the staff is absent for $|N| \cdot (|S| - 1)$ shifts after introducing the multi-shift disruptions. For simplicity, we consider each employee at most once. Next, to

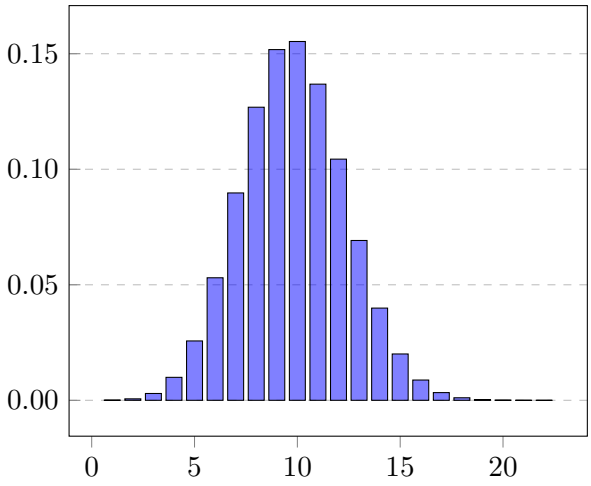


Figure 8.1: Probability mass function for the binomial distribution $B(28, 0.35)$ representing the distribution of sick leave and vacation length in days.

introduce single-shift disruptions, we randomly select $\lfloor \frac{|N| \cdot (|S|-1)}{2} \rfloor$ employee assignments (n, d, s) for $n \in N$, $d \in D$, and $s \in S \setminus \{\texttt{F}\}$ from the original schedule that are not affected by the multi-shift disruptions. These are added to the set of single-shift disruptions U^{ssd}. Since we consider a planning horizon of 28 days and four shifts (including the non-working shift) for all our schedules, the number of single- and multi-shift disruptions corresponds to about 5% ($\frac{\frac{3}{2}|N| \cdot (|S|-1)}{|N| \cdot |D| \cdot (|S|-1)}$) of the total possible working shift assignments. Hence, for each day and each shift $s \in S \setminus \{\texttt{F}\}$, on average 5% of the employees are not able to cover shift s. Finally, we need to introduce the demand disruptions R^c_{ds}, which show how the demand for employees changed compared to the original staffing

Table 8.2: Selected values for the parameters of the NRP or SRRP instances.

Parameter	Interpretation	Value
$\alpha^{\min}$	minimum number of working days in planning horizon	18
$\alpha^{\max}$	maximum number of working days in planning horizon	22
$\beta^{\min}$	minimum number of consecutive working days	2
$\beta^{\max}$	maximum number of consecutive working days	7
$\gamma_s^{\min}$ $\forall s \in S \setminus \{\mathtt{F}\}$	minimum number of assignments for shift s in the planning horizon	4
$\gamma_s^{\max}$ $\forall s \in S \setminus \{\mathtt{F}\}$	maximum number of assignments for shift s in the planning horizon	10
$\gamma_{\mathtt{F}}^{\min}$	minimum number of assignments for shift the free shift in the planning horizon	6
$\gamma_{\mathtt{F}}^{\max}$	maximum number of assignments for shift the free shift in the planning horizon	10
$\delta_s^{\min}$ $\forall s \in S \setminus \{\mathtt{N}\}$	minimum number of consecutive assignments to shift s	2
$\delta_s^{\max}$ $\forall s \in S \setminus \{\mathtt{N}\}$	maximum number of consecutive assignments to shift s	7
$\delta_{\mathtt{N}}^{\min}$	minimum number of consecutive assignments to the night shift	2
$\delta_{\mathtt{N}}^{\max}$	minimum number of consecutive assignments to the night shift	4
$\omega^{\text{SCREQ-}}$	weight for overstaffing violations	500
$\omega^{\text{SCREQ+}}$	weight for understaffing violations	1000
ω^{SPREF}	weight for preference violations	10
ω^{SEWL}	weight for uneven workloads	5
ω^{SMOD}	weight for changes in original schedule	100
$\omega^{\text{HREST}}, \omega^{\text{HWSA}}, \omega^{\text{HSTA}}, \omega^{\text{HCWSA}}, \omega^{\text{HCSTA}}$	weights for hard constraint violations in the MILP with relaxed hard constraints	100000

requirements for a day $d \in D$ and a shift $s \in S$. Therefore, we first set $R_{ds}^c = 0$ for each day $d \in D$ and each shift $s \in S$. Then, for each day $d \in D$, we select a random shift $s \in S \setminus \{\mathtt{F}\}$ and randomly choose between setting $R_{ds}^c = 1$ and $R_{ds}^c = -1$.

Constraint and Weight Parameters. Another important aspect concerning the benchmark instances are the parameters defining the constraints and the weights for soft constraint violations, which we introduced in Section 5. Table 8.2 lists these parameters, recaps their meaning, and shows the selected values for them. In this thesis, we use the same parameter values for NRP and SRRP instances. We chose the parameters such that the instances are as realistic and combinatorially hard as possible.

8.2 Computational Experiments

All algorithms were implemented in Julia[1] 1.6.1. We want to particularly mention the Flux.jl [ISF+18] package, which we used for our ML related implementations, and the MHLib.jl[2] package, which is an efficient toolbox for meta-heuristics. Furthermore, we employed Gurobi[3] 9.1.0 for solving the MILPs. Concerning the hardware, we executed all experiments in single-threaded mode on a machine with an Intel Xeon E5–2640 processor with 2.40 GHz and a memory limit of 16 GB. Since it is essential to find high-quality solutions fast in practice, we worked with a time limit of 900 seconds to evaluate our optimization algorithms. We use the optimality gap in percent to compare the performance of the approaches. For an objective value z_P of a solution, we define this gap as

$$gap[\%](z_P) = \frac{100 \cdot |z_P - z_D|}{|z_P|}, \tag{8.1}$$

which is determined with respect to the lower bound z_D obtained by solving the SRRP-MILP from Section 5.2.2 for three hours. Moreover, since all of our main approaches return feasible solutions for all instances, we will not include feasibility as a primary discussion aspect.

To enable a fair comparison, we use the same parameters $z_1 = 150$ and $z_2 = 2$ for both the randomized and the learning-based destroy operator. We observed these parameters to perform best during preliminary experimentation for the randomized LNS. Each of our destroy operators can therefore select at most 750 employee-day pairs into the destroy set. Another LNS-related configuration is that we use a time limit of five seconds for solving the sub-MILPs in the repair operator of every LNS.

Note that the choice of z_1 and z_2 also influences the parameter η determining the maximum number of employee-day pairs that our expert policy is allowed to destroy. We set this parameter to $\eta = 375$. Since the expert actions then contain fewer employee-day pairs than can be selected by the destroy operator, we increase the probability that we can include all of those pairs when applying our destroy set sampling strategy from Section 7.3. However, the primary reason we chose a lower value for η was to make the data generation process more efficient as it is faster to obtain high-quality destroy sets when solving the MILP with the local branching constraints. We always solve this local branching-based MILP using a time limit of 30 minutes. For the destroy set model, we work with a total of 150 and 50 training instances in the first and second iteration of the DAGGER algorithm, respectively. To create the training set for the temperature model, we generate trajectories for 150 randomly sampled SRRP instances.

Regarding the neural network architectures, preliminary experiments indicated that the following architecture settings perform best. For our destroy set networks, we use a single

[1] https://julialang.org
[2] https://github.com/ac-tuwien/MHLib.jl
[3] https://www.gurobi.com

GNN layer with an output dimension of 512 and an MLP with one hidden layer consisting of 256 nodes. Similarly, for the temperature networks, we also utilize a single GNN layer. However, we only require an output dimension of eight for this model. Furthermore, the associated MLP consists of one hidden layer containing four nodes. Note that we additionally use layer normalization [BKH16] between all layers in all our architectures and always apply the activation function after the layer normalization. As the activation function of our choice, we use Leaky ReLU with a slope of 0.01 for negative values. Note that we could have adjusted some design choices based on the observation that the GNNs perform best with a single layer. However, we chose to keep the proposed design as it enables other researchers and us to plug in and experiment with different GNNs.

In the following subsections, we present the results obtained from our conducted experiments. First, we compare the performance of the classical methods, including the SRRP-MILP and the randomized LNS in Section 8.2.1. Second, we describe one of our main results, which is the comparison between the randomized LNS and the LNS with the learning-based destroy operator (Section 8.2.2). Then, in Section 8.2.3, we investigate the abilities of the learning-based destroy operator to generalize to different schedules with various numbers of employees. Finally, we elaborate on the influence of the temperature model and the additional DAGGER iteration in Sections 8.2.4 and 8.2.5, respectively. As described in Section 8.1, we utilize 120 test instances consisting of random disruptions to a main schedule with $|N| = 110$ employees for most of our experiments. The only exception is Section 8.2.3, where we consider schedules with 120, 130, 140, and 150 employees for each of which we sample 30 different disruptions.

8.2.1 Comparison of Classical Methods

As classical methods, we consider solving the SRRP-MILP from Section 5.2.2 with Gurobi and the LNS applying the randomized destroy operator from Section 6.3. In the following figures and tables, we will refer to these approaches as MILP and LNS_RND, respectively.

Table 8.3 provides a summary of the performance of MILP, LNS_RND, and LNS_NN. Here, LNS_NN denotes the LNS utilizing the learning-based destroy operator. However, we will analyze its results later in Section 8.2.2. Table 8.3 contains the mean and standard deviation (SD) of the optimality gaps and objective values for the solutions of each approach. The average lower bound used to determine the optimality gaps is 834,313.15 and the associated average upper bound is 884,740.83. The bounds were computed using the SRRP-MILP with a time limit of three hours. Noticeably, LNS_RND clearly outperforms MILP by a factor greater than two on average concerning optimality gaps. LNS_RND is also more robust than MILP. While the solutions found by MILP have a standard deviation of 5.21, it is only 0.55 for LNS_RND solutions.

We complement Table 8.3 with Figure 8.2, providing a detailed comparison of the optimality gaps of the MILP and LNS_RND solutions using boxplots. For all the presented boxplots in this thesis, the y-axis represents the optimality gaps as “gap[%]”. The results for MILP are depicted by the red and the results for LNS_RND by the blue

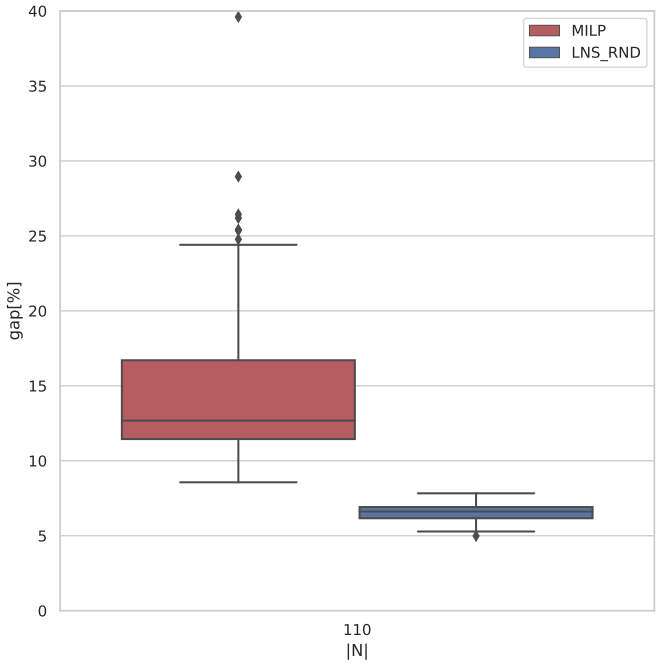


Figure 8.2: Test set optimality gaps of the MILP and LNS_RND solutions represented by boxplots.

boxplot. Even the optimality gap of the best solution found by MILP is greater than the gap of the worst solution found by LNS_RND. Again, Figure 8.2 shows that LNS_RND is more robust than MILP. Whereas the difference between the best and worst gap is about 31% for MILP, it is not even 3% for LNS_RND. Interestingly, even though some of its solutions have an optimality gap higher than 25% or 30%, MILP always found feasible solutions for the instances in our test set. Given the results presented in this section, we can conclude that the proposed classical LNS meta-heuristics is more suitable for the SRRP than directly solving its MILP.

8.2.2 Comparison of Classical LNS and Learning-Based LNS

In the previous section, we have seen that the randomized LNS (LNS_RND) comfortably outperforms the MILP approach (MILP). Therefore, we treat LNS_RND as a baseline approach to evaluate the LNS utilizing our learning-based destroy operator (LNS_NN). Note that the destroy operator of LNS_NN is the approach described in Section 7, including the additional DAGGER iteration in the data generation process and the temperature neural networks.

Again, we refer to Table 8.3, containing the main results for both LNS approaches. In addition to the mean and standard deviation values for the optimality gap and the objective value, it also contains the same statistics for the number of iterations performed

Table 8.3: Comparison of the test set results of MILP, LNS_RND, and LNS_NN.

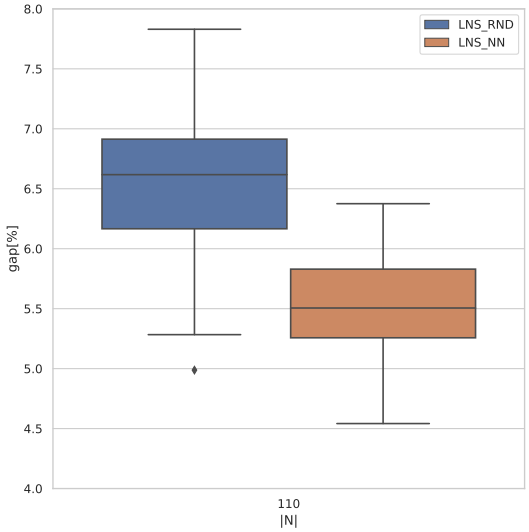
	gap[%]		obj_val		iterations	
	mean	SD	mean	SD	mean	SD
MILP	14.79	5.21	983,276.67	68,823.56	-	-
LNS_RND	6.55	0.55	892,780.00	6,377.25	154.43	42.68
LNS_NN	5.52	0.40	883,036.67	6,185.57	119.88	16.51

by the LNSs. Given Table 8.3, one can conclude that LNS_NN outperforms MILP and LNS_RND in all respects. On average, LNS_NN even surpasses the results of MILP by a factor greater than 2.65 in terms of optimality gap. Compared to LNS_RND, the average gap is slightly more than 1% lower for LNS_NN solutions. Furthermore, the mean objective value achieved by LNS_NN is about $10,000$ points lower as well. Given the instance parameters shown in Table 8.2, this is equivalent to avoiding either ten understaffing violations, 20 overstaffing violations, or 100 changes to the original schedule. From a total of 120 test instances, LNS_NN outperformed LNS_RND on 113 instances. LNS_NN achieves these results by performing, on average, about 25 iterations less than LNS_RND, which strongly indicates or even proves that our learning efforts are successful. Most probably, the decrease in iterations results from the fact that LNS_NN finds meaningful destroy sets requiring more time to be repaired.

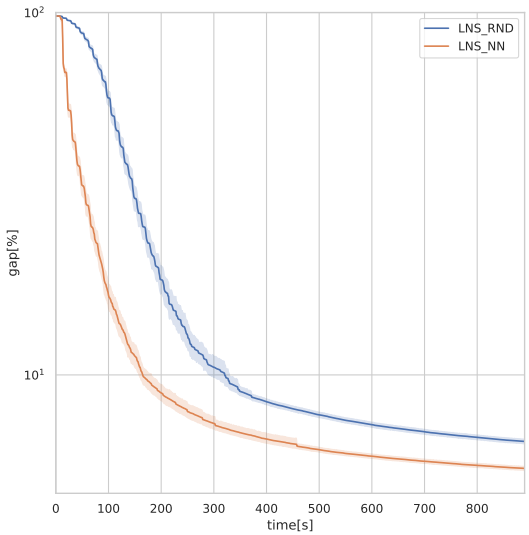
In Figures 8.3, the results of LNS_NN and LNS_RND are represented by the colors orange and blue, respectively. We provide the optimality gap results of the LNS_NN and LNS_RND solutions using boxplots in Figure 8.3a. Mainly, this representation confirms what we have already analyzed in Table 8.3. Figure 8.3b, however, offers new insights into the performance of LNS_NN and LNS_RND. It shows the average optimality gap over time. Note that the y-axis, which contains the optimality gaps, is represented in log-scale and that the region with the lighter colors around the lines depicts the standard deviation. We keep these settings for every line plot in this thesis. Figure 8.3b shows that the average performance of LNS_NN stays ahead of LNS_RND during the whole time horizon and finds better solutions significantly faster. While LNS_NN reaches an optimality gap of smaller than 10% after 165 seconds on average, it takes LNS_RND 322 seconds. Also, LNS_NN finds feasible solutions after approximately 110 seconds and LNS_RND only after 208 seconds on average.

8.2.3 Generalization to Different Instance Sizes

In this section, we investigate the generalization abilities of the learning-based destroy operator. Therefore, we use LNS_NN containing the neural network models trained on data generated from the base schedule with $|N| = 110$ employees only. We randomly create four rosters with sizes 120, 130, 140, and 150, generated using the same parameters as our base schedule. For each of these schedules, we sample 30 random disruptions such that we have 30 SRRP instances per roster size.



(a) Test set optimality gaps of the LNS_RND and LNS_NN solutions represented by boxplots.



(b) Test set average optimality gaps of the LNS_RND and LNS_NN solutions over time. Note that gap[%] is presented in log-scale.

Figure 8.3: Comparison of LNS_RND and LNS_NN based on boxplots and line plots.

Table 8.4 presents the results of the generalization experiments. It includes the average gaps and respective standard deviations, the mean objective values, and the mean iterations aggregated per instance size $|N|$. Additionally, Table 8.5 shows the average lower bounds used to compute the gaps and the associated upper bounds, both obtained with the SRRP-MILP and a time limit of three hours. In our setting, the instance size is represented by the number of employees $|N|$ as we treat the shift types and the planning horizon as fixed.

Table 8.4: Test set results of the LNS_RND and LNS_NN solutions for different instance sizes.

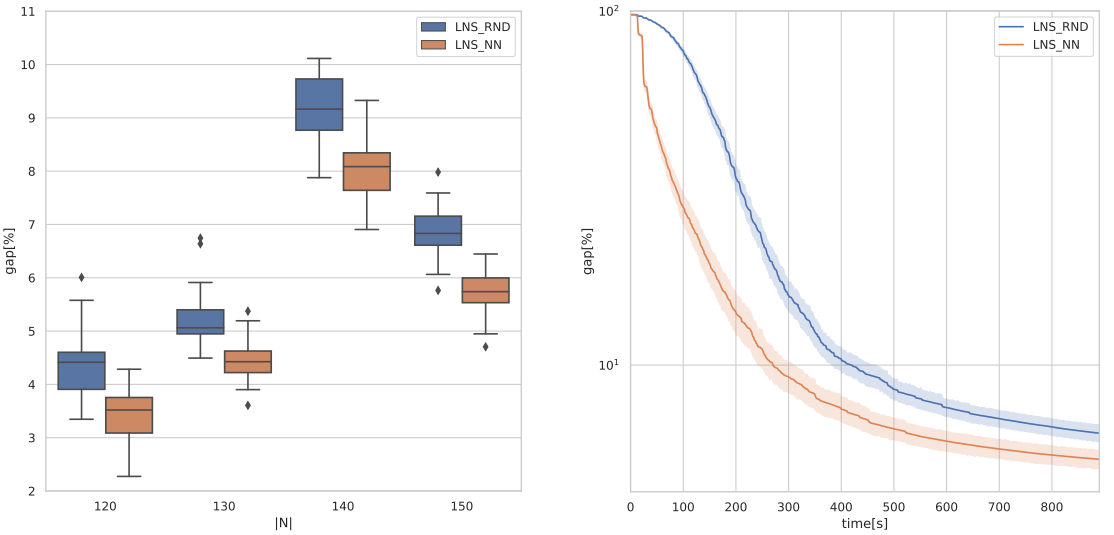
	LNS_NN				LNS_RND			
	gap[%]		obj. value	iterations	gap[%]		obj. value	iterations
$\|N\|$	mean	SD	mean	mean	mean	SD	mean	mean
120	3.42	0.53	925,543.33	118.93	4.36	0.57	934,656.67	127.27
130	4.45	0.40	1,078,910.00	113.30	5.22	0.51	1,087,716.67	128.10
140	8.04	0.60	1,313,483.33	99.77	9.20	0.58	1,330,223.33	105.43
150	5.74	0.41	1,428,643.33	95.57	6.87	0.46	1,445,943.33	100.97

We can see that LNS_NN performs better than LNS_RND in each of these instance

Table 8.5: The average upper and lower bounds for the different instance classes of the generalization experiment computed using the SRRP-MILP with a time limit of three hours.

	120	130	140	150
Upper bound	923,280.00	1,073,766.67	1,292,256.67	1,408,446.67
Lower bound	893,923.35	1,030,929.55	1,207,795.87	1,346,613.47

classes without having seen any roster different from the $|N| = 110$ main schedule during training. The results in Table 8.4 are similar to the ones from the previous section. On average, LNS_NN improves the scores of LNS_RND by around 1% in optimality gap for each instance class. In total, LNS_RND only reached better solutions than LNS_NN for two of the 120 instances from this experiment. Again, LNS_RND performs more iterations than LNS_NN on average, but the differences are lower than in the results for the test set containing only instances with $|N| = 110$. An explanation for this could be that repairing the larger schedules is more expensive in general, even with less meaningful destroy sets. Therefore, the average number of iterations decreased more drastically for LNS_RND than for LNS_NN. Figure 8.4a visually represents the described dominance of



(a) Test set optimality gaps of the LNS_RND and LNS_NN solutions represented by boxplots. In this figure, the LNS_RND and LNS_NN solutions are compared across different instance with 120, 130, 140, and 150 employees.

(b) Test set average optimality gaps of the LNS_RND and LNS_NN solutions over time. This figure shows the results aggregated for the test set consisting of schedules with 120, 130, 140, and 150 employees. Note that gap[%] is presented in log-scale.

Figure 8.4: Comparison of LNS_RND and LNS_NN for instances of different sizes based on boxplots and line plots.

LNS_NN over LNS_RND with boxplots, where the x-axis contains the different instance sizes and the y-axis shows the optimality gaps as usual. Again, the results of LNS_NN and LNS_RND are represented by the colors orange and blue, respectively.

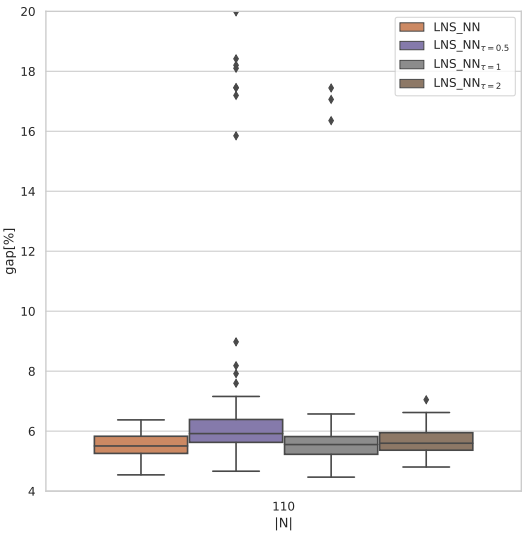
Finally, Figure 8.4b shows the average optimality gap progression of the LNS_NN (orange) and LNS_RND (blue) solutions aggregated over all instance classes. Again, the results are similar to the comparison of LNS_NN and LNS_RND for the regular test instances with $|N| = 100$. However, the standard deviation is higher for both approaches, probably resulting from the different instance sizes. Once more, LNS_NN reaches feasible solutions and solutions with an optimality gap smaller than 10% faster than LNS_RND. On average, it takes LNS_NN 190 seconds to find feasible solutions and 271 seconds to acquire solutions with a beneath 10% gap. For LNS_RND, it takes 310 and 420 seconds, respectively.

8.2.4 Influence of the Temperature Model

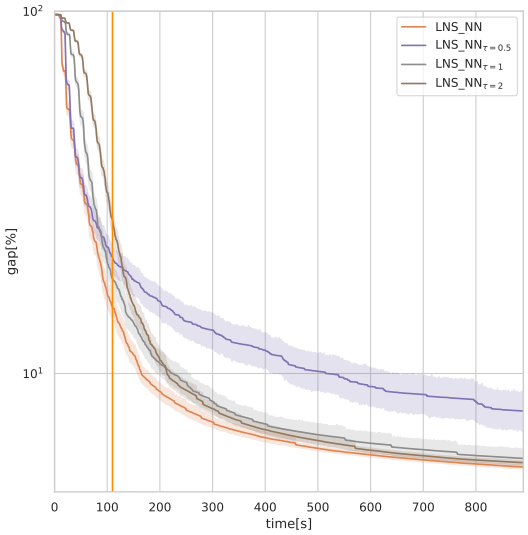
In this section, we elaborate on the influence of the temperature model π_ϕ^τ on the final performance of the LNS with the learning-based destroy operator (LNS_NN). Remember that π_ϕ^τ predicts a value τ given the output probabilities of the destroy set generation model π_θ and a current state. With this temperature parameter τ, π_ϕ^τ determines the influence of π_θ on the destroy set sampling process. Therefore, it can regulate when to trust π_θ and when to rely on a more randomized selection of the destroy set. We use the abbreviation LNS_NN$_{\tau=1}$ to refer to an LNS utilizing the learning-based destroy operator without the temperature model and using a constant temperature $\tau = 1$ instead. In this experiment, we consider LNS_NN$_{\tau=0.5}$, LNS_NN$_{\tau=1}$, and LNS_NN$_{\tau=2}$.

Figure 8.5a shows the optimality gaps for the solutions of LNS_NN$_{\tau=0.5}$, LNS_NN$_{\tau=1}$, LNS_NN$_{\tau=2}$, and LNS_NN using boxplots. The orange plot represents LNS_NN, the purple plot LNS_NN$_{\tau=0.5}$, the grey plot LNS_NN$_{\tau=1}$, and the brown plot LNS_NN$_{\tau=2}$. We cut off Figure 8.5a at a gap of 20% for better readability. LNS_NN$_{\tau=0.5}$ has seven outliers being above this mark that are not shown in this figure. Particularly noticeable, LNS_NN$_{\tau=0.5}$ and LNS_NN$_{\tau=1}$ contain several outliers. The solutions represented by these outliers are all infeasible if the associated gap is above 14%. The performance of both LNS_NN$_{\tau=2}$ and LNS_NN is very similar. Nonetheless, LNS_NN still achieves a mean gap that is about 0.15% lower on average.

For a more detailed analysis, Figure 8.4b shows the average optimality gaps of the LNS_NN$_{\tau=0.5}$, LNS_NN$_{\tau=1}$, LNS_NN$_{\tau=2}$, and LNS_NN solutions over time. Additionally, this figure contains a vertical line indicating when solutions of LNS_NN turned feasible on average. The colors representing the different approaches are the same as in Figure 8.5a. One can see that the mean performance of LNS_NN is at least as good and mostly better than all the other approaches for each point in the time horizon. In contrast to LNS_NN$_{\tau=2}$, which gives the best results on average besides LNS_NN, LNS_NN finds better solutions significantly faster. To provide some more details on the working method of the temperature model, we compare LNS_NN with LNS_NN$_{\tau=1}$



(a) Test set optimality gaps of the LNS_NN solutions and the solutions obtained with different temperatures represented by boxplots.



(b) Test set average optimality gaps of the LNS_NN solutions and the solutions obtained with different temperatures over time. The vertical line indicates when solutions of LNS_NN turned feasible on average. Note that gap[%] is presented in log-scale.

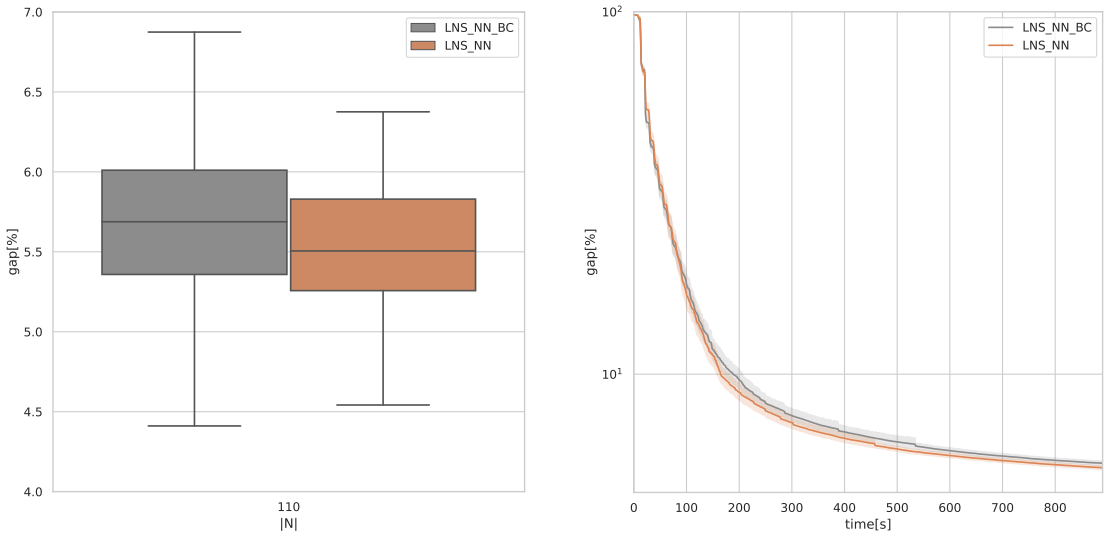
Figure 8.5: Comparison of LNS_NN$_{\tau=0.5}$, LNS_NN$_{\tau=1}$, LNS_NN$_{\tau=2}$, and LNS_NN based on boxplots and line plots.

in Figure 8.4b. We chose LNS_NN$_{\tau=1}$ since $\tau = 1$ was applied as a standard value in the data generation process. Therefore, we can use this approach as a reference to see when the temperature model detects an occasion to adapt the temperature. We observed three main behaviors of the temperature model π_ϕ^T when predicting τ. First, it completely trusts π_θ in the first few iterations of the LNS run and consistently sets $\tau = 0.1$. In the top left corner of Figure 8.4b, one can see that this choice of τ leads to faster improvements in the optimality gap since the gap between the orange line representing LNS_NN and the grey line representing LNS_NN$_{\tau=1}$ increases. Second, π_ϕ^T quickly increases τ to the highest possible value of five after these initial iterations until the solution turns feasible (orange vertical line). Therefore, the temperature model gives up some faster improvements. One can see this in Figure 8.4b as the lead of the LNS_NN line decreases compared to LNS_NN$_{\tau=1}$. However, in return for foregoing these quick improvements, the temperature model ensures that each solution turns feasible. As we have seen in 8.5a, this is not the case for LNS_NN$_{\tau=1}$. Lastly, once the solution is feasible, π_ϕ^T repeatedly selects a value for τ that is close to but below one, hence giving π_θ more influence. Again, this choice leads to faster improvements in the optimality gap. In Figure 8.4b, one can see these improvements after the vertical orange line, where the gap between the orange line (LNS_NN) and the grey line (LNS_NN$_{\tau=1}$) gradually increases.

Eventually, the average performance of LNS_NN stays ahead of LNS_NN$_{\tau=1}$ for the rest of the time horizon.

8.2.5 Influence of the DAGGER iteration

In the first approach of our data generation process, we used training data generated by sampling trajectories using the expert policy only. As we have already discussed extensively, this can lead to situations in testing where our learning-based destroy operator struggles to find adequate actions in unfamiliar states. Hence, the LNS cannot improve results as fast as desired. In this section, we investigate the influence of the DAGGER iteration on the final performance of the LNS with the learning-based destroy operator (LNS_NN). With the term DAGGER iteration, we indicate the additional iteration in the training data generation, where we employ an already trained model to sample more trajectories and obtain more relevant states for training. To elaborate on this influence, we use LNS_NN_BC being an LNS utilizing the learning-based destroy operator, including the temperature model. However, the destroy set generation models for LNS_NN_BC are trained only with the data from the expert trajectories. Hence, we did not make use of the additional DAGGER iteration for LNS_NN_BC. The BC in the name stands for behavior cloning. As in the previous sections, the destroy operator of LNS_NN contains all the features described in Section 7.



(a) Test set optimality gaps of the LNS_NN_BC and LNS_NN solutions represented by boxplots.

(b) Test set average optimality gaps of the LNS_NN_BC and LNS_NN solutions over time. Note that gap[%] is presented in log-scale.

Figure 8.6: Comparison of LNS_NN_BC and LNS_NN based on boxplots and line plots.

Figure 8.6a shows boxplots representing the test set optimality gaps of the LNS_NN_BC

(grey) and LNS_NN (orange) solutions. These boxplots indicate that LNS_NN performs slightly better than LNS_NN_BC. Whereas the maximum, third quartile, mean, and first quartile are lower for the boxplot of LNS_NN, only the minimum found optimality gap is better for LNS_NN_BC. LNS_NN improves the median optimality gap of LNS_NN_BC by approximately 0.15% and the mean gap by about 0.18%. Therefore, we can conclude that the additional DAGGER iteration positively impacts the performance of our algorithm during testing. We suspect that even more DAGGER iterations could further improve results. However, we have no data to corroborate this claim.

In Figure 8.6b, we present the average performance of LNS_NN_BC and LNS_NN over time to analyze when the positive effects of this additional training data come into effect. Once more, the results of LNS_NN and LNS_NN_BC are represented by the colors orange and grey respectively. Remember that the solutions created with LNS_NN turn feasible after about 110 seconds on average. Even though both lines in Figure 8.6b seem nearly identical, it is around this time when the average gap of LNS_NN improves faster than for LNS_NN_BC. One can see this best around 200 seconds when the orange line has its largest distance to the grey line. LNS_NN can hold this advantage until the termination of the algorithm. Hence, it appears that the additional DAGGER iteration positively influences the performance right after solutions turn feasible.

CHAPTER 9

Conclusion & Future Work

In this work, we defined a combinatorial optimization problem (COP) called the staff rerostering problem (SRRP). In contrast to already known rerostering problems such as the nurse rerostering problem (NRRP), the SRRP also contains demand disruptions indicating changes to the coverage requirements on one or more days. Also, both single- and multi-shift disruptions occur together in the SRRP making it more realistic. Furthermore, we modeled the SRRP by formulating a mixed-integer linear program (MILP) and proposed a classical large neighborhood search (LNS) meta-heuristic to solve the SRRP efficiently. To this end, we introduced a straightforward construction heuristic avoiding as many changes to an original schedule as possible and a repair operator utilizing the SRRP-MILP. For the repair operator, we proposed an additional SRRP-MILP with relaxed hard constraints dealing with the case that a current solution is infeasible. To complete the classical LNS, we suggested a randomized destroy operator incorporating the observation that it is more meaningful to destroy variables representing consecutive days.

The main contribution of this work is a learning-based LNS. More specifically, the learning-based destroy operator is controlled by a machine learning (ML) model. For this LNS, we also relied on the construction heuristic and the repair operator from the classical LNS. We trained a conditional generative model with imitation learning to generate probabilities for destroy sets and suggested a refined sampling strategy based on consecutive day selection to build high-quality destroy sets from these probabilities. We proposed a custom graph structure modeling the SRRP as an input to a graph neural network (GNN) and showed that it is possible to learn meaningful destroy sets using this representation. Existing state-of-the-art approaches employ Constraint-Variable Incidence Graphs (CVIG), resulting in huge graphs for problems like the SRRP, significantly slowing down training and inference. Therefore, we proposed this alternative approach, providing a scheme to successfully apply ML to larger instances of highly constrained COPs.

To further improve the results, we applied the Dataset Aggregation (DAGGER) imitation learning algorithm to acquire more relevant training data. Additionally, we utilized another neural network to predict suitable temperature parameters for the destroy set sampling process in a current state. In the computational results, we showed that both of these strategies positively impact the performance of our learning-based destroy operator.

Our experiments showed that the learning-based LNS outperforms the approach of solving the SRRP-MILP with Gurobi by a factor of greater than 2.65 on average in terms of optimality gap. Furthermore, the obtained results also attest strong empirical performance to the classical LNS as it comfortably outperforms the MILP-based approach. Nonetheless, we could show that the learning-based LNS still surpasses the results of the classical LNS in all respects. Most importantly, we finally showed that the proposed learning-based destroy operator, only trained on data of one schedule with different sets of disruptions, also generalizes to schedules with various numbers of employees it has never seen.

Future Work. Despite the achievements presented, there are still further potential improvements concerning our work. For example, the architecture of the utilized GNN is rather simple. It might be profitable to investigate also other GNNs variants. The generic design of our approach makes it easy to plug in and examine different architectures. As we observed in our experiments, the currently used GNN performs best with a single layer. Hence, the GNN does not take full advantage of its ability to aggregate information from larger parts of the graph. There might be more suitable GNN architectures better exploiting the given graph structure and therefore improving the performance of the learning-based destroy operator.

In the experimental results, we showed that we could successfully learn suitable temperature parameter values. In addition to this parameter, our LNS uses several different parameters, such as the destroy set size or the time allowed to repair a solution with the MILP. In future work, a further neural network, possibly with multiple heads, might be introduced to make these design choices. Eventually, the destroy operator could make its own decisions about which settings are most appropriate in the current situation.

In this thesis, we trained our neural network on a single schedule. Hence, it probably works best on this roster, and a new neural network optimally is trained for each given schedule. Considering this aspect, we designed a data generation process being as fast and lightweight as possible. However, since we observed that our proposed approach generalizes extremely well to different schedules, the next step would be to create a single neural network optimized for all possible schedules. Following this strategy, we would require training only once, meaning we could employ a more expensive data generation process, including different rosters. Therefore, the neural network could profit from more data and further improve results across various instances.

During preliminary tests, we ruled out reinforcement learning (RL) as a candidate learning algorithm since executing an episode was too expensive. Finding ways to apply RL for

learning a meaningful destroy operator is an interesting research direction concerning future work. By utilizing RL, we could directly optimize a final performance metric such as the optimality gap instead of searching for the most suitable action in a current state. Even though the strength of the LNS is the ability to escape local optima and greedily taking the locally best option might not be as harmful as in other algorithms, a policy learned by RL might unveil more intelligent strategies.

As a final point, we want to emphasize that the general approach proposed in this thesis is not limited to the SRRP but also may be applied to different classes of COPs.

List of Figures

List of Tables

List of Algorithms

Acronyms

A3C asynchronous advantage actor-critic 12

ALNS adaptive large neighborhood search 10

B&B branch-and-bound 14, 20, 24

CNN convolutional neural network 29, 30, 53

ConvGNN convolutional GNN 13, 14, 16, 30, 55

COP combinatorial optimization problem 1, 3, 9, 10, 12–14, 19–21, 23, 50, 81, 83

CVIG Constraint-Variable Incidence Graph 16, 17, 53, 81

DAGGER Dataset Aggregation 59, 60, 70–72, 78, 79, 82, 89

DCD day coverage distribution 66, 67

DPD day preference distribution 66, 67

DQN deep Q-network 13, 14

FLP facility location problem 10

GAE graph autoencoder 13, 30

GAT graph attention network 54

GCN graph convolutional network 14, 17, 54

GIN graph isomorphism network 14, 54

GNN graph neural network xi, xiii, 3, 13, 14, 19, 30, 50, 53–57, 71, 81, 82

ILP integer linear program 23, 24

LNS large neighborhood search xi, xiii, 1–4, 9, 10, 14–17, 19, 21–23, 41–43, 47, 50, 51, 53, 59, 60, 65, 70–73, 76–78, 81–83

LP linear program 23, 24

LSTM long short-term memory model 12

MAXCLIQUE maximum clique problem 14

MAXCUT maximum cut problem 13–15

MCTS Monte Carlo Tree Search 14

MDP Markov decision process 26, 49, 51

MILP mixed-integer linear program xi, xiii, 3, 4, 10–12, 15–17, 23, 24, 33, 36, 37, 39, 43, 44, 53, 58, 59, 65, 68–72, 74, 75, 81, 82, 87

ML machine learning xi, xiii, 1–4, 9, 12–16, 19, 25, 29, 49, 53, 61, 70, 81

MLP multi-layer perceptron 28, 29, 31, 32, 53, 56, 57, 71, 85

MPNN message-passing neural network 16, 30, 31

MVC minimum vertex cover problem 13–15

NN4G neural network for graphs 55

NPD nurse preference distribution 66, 67

NRP nurse rostering problem 2, 5–7, 9–12, 34, 66, 67, 69, 87

NRRP nurse rerostering problem 2, 5, 7, 11, 12, 41, 81

R-GCN relational graph convolutional network 54

RCPSP resource-constrained project scheduling problem 10

RecGNN recurrent GNN 13, 30

RL reinforcement learning 12–16, 19, 25–27, 51, 82, 83

RNN recurrent neural network 29, 30, 53

S2V structure2vec 13, 14

SCD shift coverage distribution 66, 67

SGD stochastic gradient descent 25, 29

SPD shift-preference distribution 66, 67

SRRP staff rerostering problem xi, xiii, 1–7, 19, 20, 33–36, 39–44, 46, 47, 50–54, 59–61, 65, 69–75, 81–83, 85, 87

STGNN spatial–temporal GNN 13, 30

TCC total coverage constrainedness 66, 67

TD temporal difference 27, 28

TSP traveling salesperson problem 12, 13

VNS variable neighborhood search 10

VRP vehicle routing problem 10

Bibliography

[Ach09] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[AGCL12] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. The community structure of sat formulas. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 410–423. Springer, 2012.

[ANA20] Ravichandra Addanki, Vinod Nair, and Mohammad Alizadeh. Neural large neighborhood search. In *Learning Meets Combinatorial Algorithms at Conference on Neural Information Processing Systems*, 2020.

[AXSS19] Kenshin Abe, Zijian Xu, Issei Sato, and Masashi Sugiyama. Solving np-hard problems on graphs with extended alphago zero. *arXiv preprint arXiv:1905.11623*, 2019.

[BC14] Edmund K Burke and Tim Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71–81, 2014.

[BDCBVL04] Edmund K Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–499, 2004.

[BDCRB12] Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research*, 194(1):33–57, 2012.

[Ben00] Samy Bengio. Modeling high-dimensional discrete data. In *Advances in Neural Information Processing Systems 12: Proceedings of the 1999 Conference*, volume 1, page 400. MIT Press, 2000.

[BHB+18] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016. arXiv: 1607.06450.

[BLQ10] Edmund K Burke, Jingpeng Li, and Rong Qu. 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, 2010.

[BP93] Roberto Brunelli and Tomaso Poggio. Face recognition: Features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(10):1042–1052, 1993.

[BPL+16] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[BPW+12] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[BR16] Christian Blum and Günther R Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Springer, 2016.

[CCCR21] Félix Chalumeau, Ilan Coulon, Quentin Cappart, and Louis-Martin Rousseau. Seapearl: A constraint programming solver guided by reinforcement learning. In Peter J. Stuckey, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 392–409, Cham, 2021. Springer International Publishing.

[CCK+21] Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

[CDDC+19] Sara Ceschia, Nguyen Dang, Patrick De Causmaecker, Stefaan Haspeslagh, and Andrea Schaerf. The second international nurse rostering competition. *Annals of Operations Research*, 274(1):171–186, 2019.

[CGCL20] Mingxiang Chen, Lei Gao, Qichang Chen, and Zhixin Liu. Dynamic partial removal: A neural network heuristic for large neighborhood search. *arXiv preprint arXiv:2005.09330*, 2020.

[CLLR03] Brenda Cheang, Haibing Li, Andrew Lim, and Brian Rodrigues. Nurse rostering problems—-a bibliographic survey. *European Journal of Operational Research*, 151(3):447–460, 2003.

[Dan51] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, 13:339–347, 1951.

[DDS16] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, pages 2702–2711, 2016.

[EJKS04] Andreas T Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, 2004.

[ESM+18] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1407–1416. PMLR, 2018.

[FL03] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical programming*, 98(1):23–47, 2003.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GCF+19] Maxime Gasse, Didier Chetelat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[Gér19] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2019.

[GMS05] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[GTS17] Rafael AM Gomes, Túlio AM Toffolo, and Haroldo Gambini Santos. Variable neighborhood search accelerated column generation for the nurse rostering problem. *Electronic Notes in Discrete Mathematics*, 58:31–38, 2017.

[HDCSS14] Stefaan Haspeslagh, Patrick De Causmaecker, Andrea Schaerf, and Martin Stølevik. The first international nurse rostering competition 2010. *Annals of Operations Research*, 218(1):221–236, 2014.

[HDIE14] He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. *Advances in Neural Information Processing Systems*, 27:3293–3301, 2014.

[How60] Ronald A Howard. Dynamic programming and markov processes. 1960.

[HPD19] Jiayi Huang, Mostofa Patwary, and Gregory Diamos. Coloring big graphs with alphago zero. *arXiv preprint arXiv:1902.10162*, 2019.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HT19] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*, 2019.

[ISF+18] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018.

[JORR20] Thomas Jatschka, Fabio F Oberweger, Tobias Rodemann, and Gunther R Raidl. Distributing battery swapping stations for electric scooters in an urban area. In *International Conference on Optimization and Applications*, pages 150–165. Springer, 2020.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311, 1984.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. arXiv: 1412.6980.

[KBS+16] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to Branch in Mixed Integer Programming. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 724–731, Phoenix, Arizona, February 2016. AAAI Press.

[KDZ+17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[Kha80] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[KW17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*. OpenReview.net, 2017.

[LCK18] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.

[LOR19] Antoine Legrain, Jérémy Omer, and Samuel Rosat. A rotation-based branch-and-price approach for the nurse scheduling problem. *Mathematical Programming Computation*, pages 1–34, 2019.

[M+97] Tom M Mitchell et al. Machine learning. 1997.

[MBHSL19] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2156–2167, 2019.

[MBM+16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.

[MDM+20] Akash Mittal, Anuj Dhawan, Sahil Manchanda, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 20000–20011. Curran Associates, Inc., 2020.

[Mic09] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MP03] Margarida Moz and Margarida Vaz Pato. An integer multicommodity flow model applied to the rerostering of nurse schedules. *Annals of Operations Research*, 119(1):285–301, 2003.

[MP04] Margarida Moz and Margarida Vaz Pato. Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. *Annals of Operations Research*, 128(1):179–197, 2004.

[MP07] Margarida Moz and Margarida Vaz Pato. A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research*, 34(3):667–691, 2007.

[Mul09] Laurent Flindt Muller. An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In *Proceedings of the VIII Metaheuristics International Conference (MIC)*, 2009.

[Mur12] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[MV10] Broos Maenhout and Mario Vanhoucke. Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, 13(1):77–93, 2010.

[MV11] Broos Maenhout and Mario Vanhoucke. An evolutionary approach for the nurse rerostering problem. *Computers & Operations Research*, 38(10):1400–1411, 2011.

[MV13] Broos Maenhout and Mario Vanhoucke. Reconstructing nurse schedules: Computational insights in the problem size parameters. *Omega*, 41(5):903–918, 2013.

[NBG+20] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[NGG18] Renato Negrinho, Matthew R. Gormley, and Geoffrey J. Gordon. Learning beam search policies via imitation learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10675–10684, Red Hook, NY, USA, 2018. Curran Associates Inc.

[NOST18] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9861–9871, 2018.

[Pap81] Christos H Papadimitriou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[PM08] Margarida Vaz Pato and Margarida Moz. Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic. *Journal of Heuristics*, 14(4):359–374, 2008.

[Pom88] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, page 305–313, Cambridge, MA, USA, 1988. MIT Press.

[PR10] David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

[PS98] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[RB10] Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[Ros57] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton (Project Para)*. Cornell Aeronautical Laboratory, 1957.

[SAKB19] Arslan Ali Syed, Karim Akhnoukh, Bernd Kaltenhaeuser, and Klaus Bogenberger. Neural network based large neighborhood search algorithm for ride hailing services. In *EPIA Conference on Artificial Intelligence*, pages 584–595. Springer, 2019.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Sch98] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[SGT+08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998.

[SHS+18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan

Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[SKB+18] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[SLYD20] Jialin Song, Ravi Lanka, Yisong Yue, and Bistra Dilkina. A general large neighborhood search framework for solving integer linear programs. In *Advances in Neural Information Processing Systems*, volume 33, pages 20012–20023. Curran Associates, Inc., 2020.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Sta] Statistics Austria. Sick leave. `https://www.statistik.at/web_en/statistics/PeopleSociety/health/health_status/sick_leave/index.html`, accessed 2021-04-23.

[STGR16] Haroldo G Santos, Túlio AM Toffolo, Rafael AM Gomes, and Sabir Ribas. Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, 239(1):225–251, 2016.

[SWK+21] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[VdBBDB+13] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

[VFJ15] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700. Curran Associates, Inc., 2015.

[VGG+12] Christos Valouxis, Christos Gogos, George Goulas, Panayiotis Alefragis, and Efthymios Housos. A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, 219(2):425–433, 2012.

[VM09] Mario Vanhoucke and Broos Maenhout. On the characterization and generation of nurse scheduling problem instances. *European Journal of Operational Research*, 196(2):457–467, 2009.

[WD92] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[WN99] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[WPC+20] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[WSB19] Toni I Wickert, Pieter Smet, and Greet Vanden Berghe. The nurse rerostering problem: Strategies for reconstructing disrupted schedules. *Computers & Operations Research*, 104:319–337, 2019.

[XHLJ19] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*. OpenReview.net, 2019.

[ZB20] Alexander Zai and Brandon Brown. *Deep reinforcement learning in action*. Manning Publications, 2020.

Appendix

Here, we provide the final objective values for all instances used in the main experiments. Table 1 shows the objective values of LNS_NN, LNS_RND, and MILP solutions for the test instance set with $|N| = 110$. Table 2 contains the objective values of LNS_NN and LNS_RND solutions for the test set instances from the generalization experiment with 120, 130, 140, and 150 employees. Both tables also include the lower and upper bounds used for computing the optimality gaps.

Table 1: Final objective values of LNS_NN, LNS_RND, and MILP for all test set instances with $|N| = 110$. Here, LB and UB show the lower and upper bounds computed using the SRRP-MILP with a time limit of three hours.

instance	LB	LNS_NN	LNS_RND	MILP	UB
n110_d28_s4_001	836,977	881,100	891,300	1,030,800	884,100
n110_d28_s4_002	845,615	891,600	907,100	1,061,100	897,300
n110_d28_s4_003	828,476	880,800	888,800	1,110,900	879,300
n110_d28_s4_004	827,067	872,200	887,600	941,600	864,500
n110_d28_s4_005	837,977	883,400	904,800	949,400	897,000
n110_d28_s4_006	841,662	893,400	900,400	985,300	880,400
n110_d28_s4_007	832,633	883,300	899,600	962,100	883,900
n110_d28_s4_008	838,547	881,700	895,700	1,007,100	889,800
n110_d28_s4_009	826,775	874,200	887,500	1,099,000	876,100
n110_d28_s4_010	840,414	885,100	902,100	1,138,600	881,400
n110_d28_s4_011	827,060	871,700	886,300	933,600	862,600
n110_d28_s4_012	835,372	882,400	892,000	947,900	888,600
n110_d28_s4_013	843,007	890,900	896,300	958,400	893,300
n110_d28_s4_014	828,428	870,900	891,600	1,045,800	888,300
n110_d28_s4_015	841,535	889,200	902,900	945,900	886,200
n110_d28_s4_016	831,171	881,900	892,900	945,500	890,000
n110_d28_s4_017	832,300	885,300	896,400	938,900	889,400
n110_d28_s4_018	830,546	876,400	889,600	1,076,200	877,400
n110_d28_s4_019	838,959	882,600	893,000	942,900	892,700
n110_d28_s4_020	834,389	886,200	890,900	947,300	894,500
n110_d28_s4_021	842,200	889,800	898,800	946,300	886,900
n110_d28_s4_022	839,413	880,700	891,800	930,800	883,500
n110_d28_s4_023	839,038	887,900	897,700	941,600	896,600
n110_d28_s4_024	832,375	882,400	889,600	945,300	867,600
n110_d28_s4_025	824,559	875,600	880,900	931,300	875,700
n110_d28_s4_026	838,237	888,100	900,000	944,700	899,300
n110_d28_s4_027	827,820	876,300	890,700	956,100	872,400
n110_d28_s4_028	841,308	893,400	901,900	944,900	888,300
n110_d28_s4_029	837,986	886,500	897,500	968,800	883,300
n110_d28_s4_030	818,561	870,100	888,100	935,400	868,500
n110_d28_s4_031	832,068	880,000	885,900	935,900	876,600
n110_d28_s4_032	836,866	888,800	903,400	961,500	905,700
n110_d28_s4_033	830,311	882,100	880,300	937,100	880,500
n110_d28_s4_034	836,919	887,500	884,000	1,178,000	893,000
n110_d28_s4_035	845,903	890,500	894,700	950,900	895,800
n110_d28_s4_036	835,160	881,200	884,400	980,400	878,100
n110_d28_s4_037	825,059	877,500	888,000	1,090,200	870,500
n110_d28_s4_038	837,528	887,500	895,500	967,000	873,700
n110_d28_s4_039	836,347	885,900	891,400	982,500	874,200
n110_d28_s4_040	823,882	870,900	887,600	961,100	887,900
n110_d28_s4_041	830,090	874,100	889,400	945,100	875,900
n110_d28_s4_042	831,417	882,100	893,500	947,300	879,900
n110_d28_s4_043	827,853	877,100	890,000	924,300	885,500
n110_d28_s4_044	830,173	877,000	890,800	924,900	882,600
n110_d28_s4_045	833,060	878,500	891,600	914,500	875,800
n110_d28_s4_046	826,849	876,200	891,300	964,200	866,900
n110_d28_s4_047	836,867	888,900	897,900	962,200	896,100
n110_d28_s4_048	840,569	887,900	902,200	919,300	887,800
n110_d28_s4_049	823,731	872,700	884,800	946,500	875,100
n110_d28_s4_050	836,791	885,400	893,400	934,300	885,400
n110_d28_s4_051	837,611	885,700	898,500	953,400	892,500
n110_d28_s4_052	829,008	881,200	894,100	953,700	880,800
n110_d28_s4_053	841,255	888,600	904,500	967,300	890,500
n110_d28_s4_054	835,137	885,800	892,600	947,100	887,500
n110_d28_s4_055	832,610	880,600	886,600	944,800	883,500
n110_d28_s4_056	823,376	876,900	884,000	941,000	884,400
n110_d28_s4_057	829,890	878,900	895,100	929,100	885,500
n110_d28_s4_058	837,787	891,600	898,400	968,700	906,500
n110_d28_s4_059	828,364	883,500	893,500	940,800	883,400
n110_d28_s4_060	826,232	878,800	882,600	943,000	880,000

instance	LB	LNS_NN	LNS_RND	MILP	UB
n110_d28_s4_061	840,223	889,200	898,100	950,400	893,000
n110_d28_s4_062	830,009	882,100	888,700	942,400	884,400
n110_d28_s4_063	838,773	893,900	893,400	954,100	899,200
n110_d28_s4_064	831,799	882,600	897,600	928,400	888,200
n110_d28_s4_065	843,891	898,300	895,900	961,400	888,400
n110_d28_s4_066	842,788	886,300	889,800	957,800	887,800
n110_d28_s4_067	833,044	886,900	887,700	968,000	886,500
n110_d28_s4_068	831,374	883,900	887,300	968,200	880,600
n110_d28_s4_069	840,087	891,100	899,300	952,200	890,300
n110_d28_s4_070	838,611	884,500	891,800	961,000	892,900
n110_d28_s4_071	826,618	880,000	889,200	928,800	870,400
n110_d28_s4_072	840,714	888,600	902,400	957,300	879,500
n110_d28_s4_073	835,539	883,400	897,600	958,800	872,900
n110_d28_s4_074	834,107	882,000	893,800	981,200	878,100
n110_d28_s4_075	834,773	874,800	892,300	948,000	875,300
n110_d28_s4_076	836,086	880,300	898,000	984,500	882,600
n110_d28_s4_077	825,568	879,500	886,500	972,800	876,400
n110_d28_s4_078	822,717	868,000	882,500	958,200	871,800
n110_d28_s4_079	823,406	875,200	888,500	988,400	887,100
n110_d28_s4_080	836,390	880,500	896,200	1,005,900	879,300
n110_d28_s4_081	833,410	880,000	889,400	955,000	885,100
n110_d28_s4_082	832,394	880,700	887,400	1,065,300	881,400
n110_d28_s4_083	833,509	885,100	897,300	940,400	892,600
n110_d28_s4_084	833,126	883,800	897,500	978,700	894,200
n110_d28_s4_085	836,030	885,600	891,000	926,300	883,100
n110_d28_s4_086	848,270	891,300	901,000	970,900	894,800
n110_d28_s4_087	837,428	884,000	890,200	963,000	892,400
n110_d28_s4_088	828,210	875,900	881,900	1,072,400	877,900
n110_d28_s4_089	834,937	882,800	882,800	1,039,100	885,500
n110_d28_s4_090	833,010	884,000	897,700	1,027,900	882,800
n110_d28_s4_091	839,062	885,700	903,100	976,000	893,000
n110_d28_s4_092	832,538	886,400	892,500	1,077,900	888,800
n110_d28_s4_093	840,127	881,600	895,900	1,010,800	889,400
n110_d28_s4_094	831,603	874,500	889,600	1,066,500	867,700
n110_d28_s4_095	829,362	873,800	885,300	963,200	882,300
n110_d28_s4_096	841,785	888,800	904,100	1,059,500	893,900
n110_d28_s4_097	832,662	885,000	882,300	1,059,200	886,000
n110_d28_s4_098	838,065	883,600	887,200	1,087,700	894,700
n110_d28_s4_099	837,292	886,800	889,900	1,043,900	883,500
n110_d28_s4_100	835,315	883,600	889,900	1,061,300	887,500
n110_d28_s4_101	828,838	882,500	891,200	1,096,400	882,400
n110_d28_s4_102	842,659	894,200	905,200	1,145,400	899,700
n110_d28_s4_103	824,454	873,100	885,100	1,089,300	878,200
n110_d28_s4_104	837,231	885,000	894,800	1,386,200	894,000
n110_d28_s4_105	836,159	887,600	898,600	1,067,700	890,100
n110_d28_s4_106	843,629	891,000	900,500	1,064,300	899,200
n110_d28_s4_107	835,446	882,100	888,700	1,119,200	899,200
n110_d28_s4_108	844,868	890,600	896,900	951,100	888,400
n110_d28_s4_109	836,741	885,900	888,500	947,400	893,000
n110_d28_s4_110	836,178	884,700	889,800	936,400	878,800
n110_d28_s4_111	843,662	893,400	911,000	942,400	882,400
n110_d28_s4_112	829,080	883,000	892,000	922,200	872,900
n110_d28_s4_113	819,905	870,200	884,000	922,400	874,100
n110_d28_s4_114	834,685	874,400	882,400	924,300	881,400
n110_d28_s4_115	841,497	898,800	899,600	983,300	891,700
n110_d28_s4_116	842,284	883,700	886,500	953,700	889,800
n110_d28_s4_117	832,509	887,900	887,000	943,500	888,200
n110_d28_s4_118	829,857	881,600	886,700	926,900	875,100
n110_d28_s4_119	832,398	877,700	890,300	945,500	882,900
n110_d28_s4_120	831,666	884,400	898,400	923,200	885,600

Table 2: Final objective values of LNS_NN and LNS_RND for all test set instances with 120, 130, 140, and 150 employees. Here, LB and UB show the lower and upper bounds computed using the SRRP-MILP with a time limit of three hours.

instance	LB	LNS_NN	LNS_RND	UB
n120_d28_s4_001	908,585	938,100	940,900	936,300
n120_d28_s4_002	891,877	920,200	931,900	920,600
n120_d28_s4_003	894,689	928,200	936,100	921,600
n120_d28_s4_004	895,437	929,500	937,900	925,900
n120_d28_s4_005	896,763	925,600	938,100	929,000
n120_d28_s4_006	887,270	914,400	922,600	914,900
n120_d28_s4_007	904,905	937,100	946,100	935,400
n120_d28_s4_008	895,066	928,300	937,100	934,600
n120_d28_s4_009	898,403	932,300	944,500	933,600
n120_d28_s4_010	896,857	923,100	932,000	919,100
n120_d28_s4_011	899,074	922,700	941,100	920,900
n120_d28_s4_012	882,820	921,700	927,100	919,700
n120_d28_s4_013	885,496	922,900	926,200	924,800
n120_d28_s4_014	898,291	938,500	955,700	942,000
n120_d28_s4_015	891,114	923,800	926,400	909,300
n120_d28_s4_016	896,813	933,900	940,300	916,800
n120_d28_s4_017	892,957	926,900	936,800	924,100
n120_d28_s4_018	886,918	921,800	926,300	923,200
n120_d28_s4_019	883,635	921,300	925,600	919,400
n120_d28_s4_020	892,769	930,500	945,500	929,000
n120_d28_s4_021	905,283	931,900	938,000	930,500
n120_d28_s4_022	887,118	919,300	928,200	904,300
n120_d28_s4_023	899,668	920,600	934,700	923,400
n120_d28_s4_024	902,259	933,100	946,500	934,500
n120_d28_s4_025	901,382	929,700	936,800	926,300
n120_d28_s4_026	883,843	917,400	922,000	915,900
n120_d28_s4_027	888,638	909,400	919,400	907,700
n120_d28_s4_028	886,783	921,700	928,600	919,900
n120_d28_s4_029	887,398	916,400	926,900	910,900
n120_d28_s4_030	895,576	926,000	940,400	924,800
n130_d28_s4_001	1,031,024	1,084,800	1,095,800	1,076,900
n130_d28_s4_002	1,018,064	1,067,100	1,077,600	1,061,700
n130_d28_s4_003	1,039,323	1,088,100	1,097,700	1,092,300
n130_d28_s4_004	1,039,321	1,084,100	1,094,200	1,081,500
n130_d28_s4_005	1,049,172	1,100,000	1,100,200	1,086,100
n130_d28_s4_006	1,026,743	1,072,200	1,082,300	1,068,300
n130_d28_s4_007	1,021,163	1,066,100	1,074,600	1,061,500
n130_d28_s4_008	1,035,598	1,084,200	1,089,800	1,081,000
n130_d28_s4_009	1,038,726	1,080,900	1,087,600	1,076,400
n130_d28_s4_010	1,025,681	1,074,700	1,098,600	1,074,800
n130_d28_s4_011	1,037,393	1,084,200	1,096,800	1,083,100
n130_d28_s4_012	1,033,056	1,075,400	1,088,400	1,068,200
n130_d28_s4_013	1,028,619	1,075,600	1,079,900	1,076,300
n130_d28_s4_014	1,031,631	1,080,800	1,092,800	1,075,400
n130_d28_s4_015	1,033,861	1,077,500	1,088,200	1,079,300
n130_d28_s4_016	1,032,003	1,090,600	1,091,900	1,075,900
n130_d28_s4_017	1,030,168	1,076,100	1,083,700	1,067,200
n130_d28_s4_018	1,026,535	1,073,100	1,081,000	1,070,200
n130_d28_s4_019	1,033,572	1,079,800	1,087,700	1,074,400
n130_d28_s4_020	1,030,686	1,081,900	1,086,500	1,071,400
n130_d28_s4_021	1,025,034	1,076,600	1,084,900	1,066,300
n130_d28_s4_022	1,021,755	1,064,400	1,077,900	1,068,000
n130_d28_s4_023	1,020,645	1,067,100	1,071,000	1,059,200
n130_d28_s4_024	1,031,785	1,088,300	1,106,400	1,075,000
n130_d28_s4_025	1,029,265	1,080,600	1,087,400	1,076,300
n130_d28_s4_026	1,032,140	1,075,000	1,084,900	1,072,800
n130_d28_s4_027	1,015,581	1,069,900	1,068,100	1,059,000
n130_d28_s4_028	1,033,095	1,082,800	1,087,000	1,077,600
n130_d28_s4_029	1,041,540	1,080,500	1,095,700	1,080,100
n130_d28_s4_030	1,034,697	1,084,900	1,092,900	1,076,800

instance	LB	LNS_NN	LNS_RND	UB
n140_d28_s4_001	1,213,462	1,309,200	1,327,000	1,301,300
n140_d28_s4_002	1,202,935	1,307,700	1,312,700	1,285,600
n140_d28_s4_003	1,217,304	1,307,600	1,321,400	1,305,700
n140_d28_s4_004	1,202,921	1,293,400	1,319,600	1,266,900
n140_d28_s4_005	1,201,224	1,309,900	1,335,500	1,295,400
n140_d28_s4_006	1,218,826	1,319,100	1,338,200	1,289,900
n140_d28_s4_007	1,211,185	1,318,000	1,330,200	1,287,400
n140_d28_s4_008	1,217,002	1,317,700	1,333,600	1,299,800
n140_d28_s4_009	1,202,109	1,319,200	1,328,600	1,294,400
n140_d28_s4_010	1,209,845	1,320,200	1,341,600	1,297,700
n140_d28_s4_011	1,203,715	1,295,700	1,332,600	1,284,100
n140_d28_s4_012	1,200,832	1,304,600	1,324,200	1,285,000
n140_d28_s4_013	1,202,397	1,318,600	1,332,800	1,281,600
n140_d28_s4_014	1,209,162	1,323,600	1,333,100	1,295,300
n140_d28_s4_015	1,203,839	1,314,500	1,335,300	1,297,800
n140_d28_s4_016	1,206,821	1,315,000	1,321,200	1,292,100
n140_d28_s4_017	1,201,992	1,303,800	1,331,900	1,290,800
n140_d28_s4_018	1,218,094	1,325,000	1,336,500	1,301,800
n140_d28_s4_019	1,207,887	1,313,100	1,343,800	1,295,600
n140_d28_s4_020	1,198,602	1,305,300	1,332,600	1,284,400
n140_d28_s4_021	1,205,402	1,305,400	1,335,600	1,294,600
n140_d28_s4_022	1,202,511	1,326,200	1,329,400	1,283,500
n140_d28_s4_023	1,199,900	1,310,500	1,326,000	1,288,000
n140_d28_s4_024	1,211,903	1,312,100	1,325,000	1,288,300
n140_d28_s4_025	1,219,203	1,328,500	1,334,500	1,300,900
n140_d28_s4_026	1,214,651	1,322,100	1,334,900	1,294,800
n140_d28_s4_027	1,205,955	1,305,700	1,319,800	1,279,800
n140_d28_s4_028	1,211,864	1,318,700	1,330,100	1,311,400
n140_d28_s4_029	1,198,805	1,321,900	1,317,800	1,299,800
n140_d28_s4_030	1,213,513	1,312,200	1,341,200	1,294,000
n150_d28_s4_001	1,341,288	1,423,300	1,443,300	1,406,700
n150_d28_s4_002	1,343,609	1,422,100	1,444,200	1,408,100
n150_d28_s4_003	1,356,149	1,423,100	1,450,400	1,415,900
n150_d28_s4_004	1,343,066	1,435,600	1,452,200	1,406,500
n150_d28_s4_005	1,343,199	1,426,800	1,429,900	1,409,000
n150_d28_s4_006	1,329,937	1,420,500	1,445,300	1,402,000
n150_d28_s4_007	1,341,877	1,427,500	1,440,000	1,408,500
n150_d28_s4_008	1,344,134	1,414,100	1,439,100	1,391,300
n150_d28_s4_009	1,351,095	1,431,200	1,447,700	1,409,300
n150_d28_s4_010	1,348,715	1,430,500	1,447,900	1,410,200
n150_d28_s4_011	1,345,292	1,432,900	1,449,500	1,401,200
n150_d28_s4_012	1,357,296	1,438,200	1,440,300	1,421,700
n150_d28_s4_013	1,352,408	1,433,200	1,453,800	1,411,700
n150_d28_s4_014	1,338,590	1,419,400	1,433,900	1,402,400
n150_d28_s4_015	1,348,719	1,437,400	1,459,500	1,413,900
n150_d28_s4_016	1,342,865	1,422,100	1,446,900	1,405,700
n150_d28_s4_017	1,345,604	1,421,800	1,443,100	1,406,800
n150_d28_s4_018	1,347,928	1,426,100	1,449,900	1,403,800
n150_d28_s4_019	1,349,459	1,423,100	1,447,000	1,407,300
n150_d28_s4_020	1,354,592	1,438,000	1,445,600	1,421,100
n150_d28_s4_021	1,347,892	1,425,500	1,440,700	1,399,100
n150_d28_s4_022	1,340,256	1,427,300	1,443,700	1,399,600
n150_d28_s4_023	1,350,645	1,441,800	1,454,300	1,418,000
n150_d28_s4_024	1,344,146	1,423,900	1,438,500	1,406,600
n150_d28_s4_025	1,348,148	1,434,100	1,453,300	1,407,400
n150_d28_s4_026	1,357,736	1,432,800	1,455,500	1,424,100
n150_d28_s4_027	1,355,590	1,440,000	1,452,900	1,416,500
n150_d28_s4_028	1,341,801	1,427,300	1,450,100	1,406,500
n150_d28_s4_029	1,345,753	1,435,200	1,445,900	1,410,200
n150_d28_s4_030	1,340,598	1,424,500	1,433,900	1,402,300