Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Sigurd Sigurdsson Langfeldt, Magnus Hauge Langholm, Tomas Haugland Spangelo

Neural Network Assisted Large Neighborhood Search for Personnel Rostering

Master's thesis in Industrial Economics and Technology Management
Supervisor: Anders Nordby Gullhav
Co-supervisor: Henrik Andersson
June 2022

NTNU
Kunnskap for en bedre verden

Sigurd Sigurdsson Langfeldt, Magnus Hauge Langholm, Tomas Haugland Spangelo

Neural Network Assisted Large Neighborhood Search for Personnel Rostering

Master's thesis in Industrial Economics and Technology Management
Supervisor: Anders Nordby Gullhav
Co-supervisor: Henrik Andersson
June 2022

Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

NTNU

Kunnskap for en bedre verden

NTNU

Kunnskap for en bedre verden

Master's Thesis

Managerial Economics and Operations Research

TIØ4905

Neural Network Assisted Large Neighborhood Search for Personnel Rostering

Authors:
Sigurd Sigurdsson Langfeldt
Magnus Hauge Langholm
Tomas Haugland Spangelo

Supervisor:
Assoc. Prof. Anders Nordby Gullhav
Co-supervisor:
Prof. Henrik Andersson

Department of Industrial Economics
and Technology Management

June 9, 2022

Preface

This master's thesis concludes our Master of Science degree in Industrial Economics and Technology Management at the Norwegian University of Science and Technology (NTNU), specializing in Managerial Economics and Operations Research. The master's programme is a part of the Department of Industrial Economics and Technology Management and combines technology, management, and economics subjects. In addition to specializing in Managerial Economics and Operations Research, we have a background in Computer Science, especially Artificial Intelligence, from the master's programme. We started the work for this master's thesis in January 2022 and concluded the work in June 2022. This thesis was written in collaboration with Visma Resolve and based on the work with our specialization project for the fall semester of 2021 in Langfeldt et al. (2021).

Firstly, we want to thank our supervisor, Anders Nordby Gullhav, and co-supervisor, Henrik Andersson, for your guidance and invaluable feedback on our work. Furthermore, we want to thank you for the freedom to fully take advantage of our interdisciplinary background in Operations Research and Artificial Intelligence, which allowed us to explore an exciting solution approach to combinatorial optimization problems. We would also like to thank Martin Olstad, Jacub Jonik and Vlatka Janeš from Visma Resolve for their guidance and feedback on possible directions to explore in this thesis. Especially we would like to thank Martin Olstad for the continuous feedback we received throughout the entire process.

Finally, we would like to express our sincere gratitude to our family and friends for your unconditional support; you helped us stay motivated despite any challenges and hiccups along the way.

Sigurd Sigurdsson Langfeldt, Magnus Hauge Langholm and Tomas Haugland Spangelo
Trondheim, Norway
June 2022

Abstract

Personnel rostering is an essential task for many organizations that can be complex and time-consuming, yet it is often done manually. Our industry partner, Visma Resolve, provides software solutions to automate the shift scheduling process, which must balance generality and specificity to appeal to their diverse customers. We denote this general shift scheduling problem as the Resolve Rostering Problem (RRP). Visma Resolve's software solution uses a heuristic approach to solve their rostering problem. Due to the recent popularity and advances in machine learning technology, Visma Resolve have expressed interest in integrating such technology into their solutions. Machine learning methods are also becoming increasingly popular in operation research communities to solve combinatorial optimization problems, making it an interesting academic research field.

This thesis aims to combine techniques from operations research and machine learning to solve a general shift scheduling problem. We present a Neural Network Assisted Large Neighborhood Search (NNALNS), which integrates an Artificial Neural Network (ANN) in an Adaptive Large Neighborhood Search (ALNS). Using reinforcement learning techniques, we train the ANN to learn operator selection policies, replacing the online learning mechanism of ALNS. NNALNS uses problem-specific and search-based features as input to the ANN to provide information on the state during the search.

The results in this thesis show that NNALNS manages to outperform ALNS on all but one of the problem instances when the ANN is trained and tested on the same instance. NNALNS also exhibits an ability to learn good, generalized operator selection policies applicable to different problem instances. In a real-world use case, these results points in favor of using pre-trained policies. The RRP instances for a particular customer of Visma Resolve are often similar in structure between scheduling periods and can be solved by NNALNS without training new selection policies.

By enriching the state representation with problem-specific features, NNALNS obtains more rewards from the reinforcement learning environment. However, a possible disassociation between rewards and obtained objective values inhibits an increase in objective value.

Our main contribution is the inclusion of an enriched state representation of the RRP and the design of tailored operators for the problem. In addition, NNALNS constitutes a novel solution approach to personnel rostering problems, bridging the gap between machine learning and operations research.

Sammendrag

Allokering av arbeidskraft er en essensiell oppgave hos mange organisasjoner som kan være både kompleks og tidkrevende. Til tross for dette er den dominerende praksisen å sette opp timeplaner manuelt. Vår industripartner, Visma Resolve, leverer programvareløsninger for å automatisere skiftplanleggingsprosessen. Deres programvareløsning må balansere generalitet og spesifisitet for å appellere til mange ulike kunder. Vi kaller det generelle skiftplanleggingsproblemet til Visma Resolve for Resolves skiftsplanleggingsproblem (Resolve Rostering Problem - RRP), og Visma Resolves programvareløsning bruker heuristiske løsningsmetoder for å løse det. På grunn av den voksende populariteten og utviklingen til maskinlæring har Visma Resolve uttrykt at de ønsker å integrere maskinlæring i deres eksisterende løsning. Også innen operasjonsanalysemiljøer har maskinlæringsmetoder blitt populære for å løse kombinatoriske optimeringsproblemer, hvilket gjør det til et spennende akademisk forskningsfelt.

Denne masteroppgaven har som formål å forene teknikker fra operasjonsanalyse og maskinlæring. Vi presenterer et nevralt nettverk-assistert nabolagssøk (Neural Network Assisted Large Neighborhood Search - NNALNS) som integrerer et nevralt nettverk i et adaptivt stort nabolagssøk (Adaptive Large Neighborhood Search - ALNS). Vi trener det nevrale nettverket til å lære seleksjonsstrategier for operatorer ved å bruke teknikker fra forsterket læring, hvilket erstatter den adaptive læringsmekanismen til ALNS. NNALNS bruker problemspesifikke og søksbaserte attributter som input til det nevrale nettverket for å beskrive tilstanden i løpet av søket.

Resultatene i masteroppgaven viser at NNALNS klarer å utkonkurrere ALNS på alle instanser utenom én når det nevrale nettverket er trent og testet på den samme instansen. NNALNS viser også evne til å kunne lære gode, generelle operatorseleksjonsstrategier som kan benyttes på ulike probleminstanser. I praksis kan disse resultatene peke mot at det er fordelaktig å bruke forhåndstrente operatorseleksjonsstrategier. RRP-instansene for en spesifikk kunde av Visma Resolve er ofte like i struktur mellom planleggingsperioder og kan derfor løses av NNALNS uten å måtte trene nye nevrale nettverk.

Ved å berike tilstandsrepresentasjonen med problemspesifikke attributter så klarer NNALNS å sanke flere belønninger (rewards). Likevel kan en mulig dissosiasjon mellom belønninger og objektivverdi hemme en økning i objektivverdi.

Vårt hovedbidrag er å inkludere en problemspesifikk tilstandsrepresentasjon og implementasjonen av operatorer som er skreddersydd for RRPet. I tillegg utgjør

NNALNS en nyskapende løsningstilnærming til skiftplanleggingsproblemer og bidrar til å lukke gapet mellom maskinlæring og operasjonsanalyse.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ALNS Adaptive Large Neighborhood Search.

ANN Artificial Neural Network.

COP Combinatorial Optimization Problem.

DNN Deep Neural Network.

LNS Large Neighborhood Search.

MIP Mixed Integer Programming.

ML Machine Learning.

NN Neural Network.

NNALNS Neural Network Assisted Large Neighborhood Search.

NRP Nurse Rostering Problem.

OR Operations Research.

PPO Proximal Policy Optimization.

PRP Personnel Rostering Problem.

RL Reinforcement Learning.

RRP Resolve Rostering Problem.

ULNS Uniform Large Neighborhood Search.

Chapter 1

Introduction

Organizing labor resources is an essential part of most organizations. Especially in tertiary industries where labor costs constitute a dominant part of an organization's expenses, effective use of labor resources becomes vital to a company's ability to deliver quality products and services. In Norway, the portion of companies in service-based industries has grown from 55% to 78% over the last decade, placing labor planning at an increasingly relevant position in society (SSB, 2013, 2020). A common way of organizing labor is through shift-based work, predominantly when organizations require labor services outside regular work hours. Shift-based working arrangements often require a high degree of coordination and planning to provide continuous services. However, setting up shift schedules can be a complex and time-consuming task as several considerations must be addressed to ensure proper labor allocation. Since labor consists of individual employees with different skills, education, preferences, and agendas, this significantly complicates the process. Hence, tools that can ease this task are of high value.

A Personnel Rostering Problem (PRP) refers to a problem of producing shift schedules within a finite time horizon. In academia, Nurse Rostering Problems (NRPs), which explicitly address work time scheduling in healthcare institutions where some of the most demanding scheduling tasks are found, are prominent examples of PRPs. NRPs are well-studied problems due to their complexity and societal importance. They belong to the class of NP-hard problems, making them particularly hard to solve as the complexity grows exponentially with the problem size. Nevertheless, finding effective solution methods for NRPs can significantly impact how healthcare institutions operate and utilize their workforce.

This thesis is the final submission for our master's degree at NTNU. It is a continuance of a specialization project from the fall of 2021 (Langfeldt et al., 2021). Extensive literature research on PRPs and the problem formulation from the specialization project report are included in this thesis. Our work is conducted in collaboration with Visma Resolve, a leading provider of commercial rostering software in the Nordics.

Visma Resolve provides a complete rostering system to automate the task of constructing shift schedules. Their system needs to balance generality and specificity

to appeal to a broad set of customers from different industries with various needs and scheduling practices. We denote the general shift scheduling problem at Visma Resolve as the Resolve Rostering Problem (RRP). Although the RRP is a general shift scheduling problem, it strongly links to NRPs. We view it as an NRP variant, which allows us to draw parallels to similar problems in the literature. A large part of Visma Resolves's customer base are institutions within the healthcare industry. Hence, regarding it as an NRP variant does not cause a loss of generality or relevance.

A challenge faced at Visma Resolve is for their system to produce desirable shift schedules fast enough for their customers to see the value of transitioning from manual to automatic shift scheduling. As exact solving methods are inadequate for this purpose for the RRP, they apply heuristic solving approaches. However, they are continuously exploring ways to improve their algorithms' performance. Due to the recent advances in Machine Learning (ML) methods, Visma Resolve has expressed interest in integrating such technology with their current heuristic solution. In the last decades, ML technology has become growingly popular in many problem domains, fueled by the increased availability of data and computing power (Jordan and Mitchell, 2015). Although ML is often associated with the field of computer science, there are clear similarities between the problems ML addresses and central topics in the academic field of Operations Research (OR). Thus, integrating ML in traditional OR techniques, such as meta-heuristics, is a growing field of research (Bengio et al., 2021). The purpose of this thesis is to experiment with such an approach of combining OR and ML techniques to solve real-life rostering problems, thus utilizing the strengths of two academic disciplines.

We present a Neural Network Assisted Large Neighborhood Search (NNALNS), which integrates an Artificial Neural Network (ANN) in an Adaptive Large Neighborhood Search (ALNS) framework. Using Reinforcement Learning (RL) techniques, we train the ANN to learn operator selection strategies at different stages of the search, replacing the online learning mechanism of ALNS. NNALNS includes problem-specific features of an RRP solution and search-based features as input to the ANN to provide information on the state during the search.

Our goals for this thesis are to:

1. Investigate whether NNALNS can learn more intelligent operator selection strategies for the RRP compared to ALNS.
2. Include an enriched state representation of an RRP solution for NNALNS.
3. Examine the generalizability of learned selection strategies across different RRP instances.
4. Develop operators tailored to the RRP targeting different properties of a solution.

At Visma Resolve, the structure of shift schedules often varies between different clients. The core idea is that by using pre-trained networks, an RL agent can learn

to exploit the structures for a particular client, thus being able to handle reasonable changes to the problem.

The results in this thesis show that NNALNS manages to outperform ALNS on all but one of the problem instances when the ANN is trained and tested on the same instance. NNALNS also exhibits an ability to learn good, generalized operator selection policies applicable to different problem instances. In a real-world use case, these results points in favor of using pre-trained policies. The RRP instances for a particular customer of Visma Resolve are often similar in structure between scheduling periods and can be solved by NNALNS without training new selection policies.

The main contributions of this thesis are the development of NNALNS and combining techniques from OR and ML to solve real-life rostering problems. Using RL to learn operator selection strategies with complex destroy-repair operators, in combination with smaller hybrid operators, forms a novel solution approach to a PRP, namely the RRP. Furthermore, we include an enriched state representation including problem-specific features as input to the ANN parameterizing the operator selection strategy.

1.1 Outline

Chapter 2 provides the necessary background and theory to grasp the contents of the rest of this thesis. In Chapter 3, we provide an extensive literature review relevant to NNALNS and the RRP, before Chapter 4 presents the problem definition of the RRP. Following, Chapter 5 describes the Mixed Integer Programming (MIP) formulation of the RRP and Chapter 6 presents NNALNS and our adaptation of ALNS to the RRP. Chapter 7 describes the test instances and parameter decisions, followed by the empirical results of applying ALNS and NNALNS to these instances in Chapter 8. Lastly, Chapter 9 concludes the most important results and contributions of this master's thesis and discusses interesting directions for future research.

Chapter 2

Background and Theory

In this chapter, we present the relevant background and theory for this thesis. Firstly, Section 2.1 presents terminology that is crucial to the understanding of subsequent chapters regarding PRPs. Next, Section 2.2 provides a high-level motivation for works on the personnel rostering space, before the healthcare industry is discussed in Section 2.3. We describe our industry partner, Visma Resolve, and their problem regarding personnel rostering in Section 2.4. Lastly, Section 2.5 provides an introduction to RL concepts relevant to this thesis. Parts of this chapter are derived from our specialization project (Langfeldt et al., 2021).

2.1 Terminology

This section shows some terms used throughout this thesis, which can be useful for the reader to understand the content regarding PRPs in the subsequent sections and chapters.

- *Shift*: A shift is a defined working time for a specific day that can be assigned to one or more employees, where they can cover demand.
- *Shift Pattern*: A shift pattern is an ordered combination of shifts assigned to an employee over a number of days. Some are more or less desirable than others.
- *Shift Schedule*: A shift schedule, also referred to as a roster, is a complete plan showing which shift patterns are assigned to employees within a finite time horizon, and who covers what demand.
- *Fairness*: Fairness is a subjective term, but it generally regards the distribution of workload, generally undesirable shifts and shift patterns, and personal request approval between employees in a roster.
- *Competence*: The ability, or in some cases the permission, to perform specific tasks or cover specific types of demand.

- *Isolated (off) day*: Working a single day without working neighboring days (or opposite).
- *Demand*: The requirement that a certain number of employees work a specific time period.

2.2 Personnel Rostering

Personnel rostering is the task of creating work schedules for a set of employees within a finite time horizon. The problem is particularly important in shift-based industries where the demand for labor is spread continuously throughout the day, which requires labor outside regular working hours. This calls for more extensive workforce coordination than regular working hour arrangements as the employees can have significant variations in working times. Because some shifts and shift patterns can be undesirable for many employees, it introduces the need to create fair working schedules. Schedule planners need to balance between providing good working conditions for their employees and allocating enough labor to meet the demand to ensure high-quality services.

Presently, personnel rostering is usually done manually by the working staff. Manual workforce planning is a complex and time-consuming task that risks creating personnel allocations of inadequate quality. Poor schedules can be a source of conflicts, frustration and even be illegal in regards to domestic working regulations. The complexity grows even further when the planning process takes the personal preferences of individual employees into account. This might result in unfair schedules as more influential employees will have an advantage in the scheduling process. Perceived fairness among employees has been identified as a key factor to promote good working conditions and employee satisfaction as well as providing healthy working environments (Wolbeck, 2019).

Decision support tools for personnel rostering have been studied in academia for decades and have gained traction in many industries. There are several advantages of computerizing workforce rostering: reducing planning time, evaluating schedule allocations impartially between employees, and disallowing illegal schedules, to point out a few. Nevertheless, the rostering problem itself varies largely between industries and application areas, complicating the process of developing useful general scheduling support tools. In addition, the overall complexity of rostering problems makes even moderate roster sizes computationally heavy, even for supercomputers. However, the emergence of cloud computing technologies in the last decade has given rise to automatic scheduling systems. This is because computation power has become more available, cost-effective, and highly scalable, making it possible to create good solutions within a reasonable time frame.

2.3 The Healthcare Industry

The healthcare sector is a labor-intensive industry, where complex activity planning and coordination are central to operations, explaining the prevelence of NRPs among PRPs. During the recent Covid-19 pandemic, healthcare institutions' capacity has been under pressure, leading to stressed working conditions and long hours for many healthcare employees. Norwegian healthcare unions have recently expressed concerns about understaffing and poor working conditions among their union members, which highly affects the quality of the delivered health services as well as the working environment (Dagbladet, 2021). In addition, Norwegian healthcare institutions are already experiencing an increase in demand for health services for years. Norway has, like many other industrialized countries, an increasingly aging population. Combined with an increase in life expectancy, the demand for health services is expected to grow in the future (SSB, 2019).

Medical breakthroughs and a higher frequency of complex treatment methods have made resource planning in healthcare a complicated task. Healthcare planners need to ensure that the highly specialized workforce is used efficiently and coordinates the use of expensive and advanced medical equipment. Furthermore, stricter policies and regulations regarding waiting times for healthcare services are pushed by the government. In Denmark, for example, it has been politically decided that all patients have a right to be treated within 30 days of arrival (Range, 2021). Such policies intensify the need for efficient allocation and coordination of the workforce.

Hospitals and similar healthcare institutions are good examples of shift-based organizations. Patients need to be treated and monitored at all hours of the day. Thus, enough nurses, doctors, and other medical personnel need to be available to secure a satisfying service level. Offering such a 24-hour service is a complex task, so the health sector is an especially relevant area for personnel rostering problems.

Scheduling Practices in Healthcare Institutions

Shift scheduling practices vary between healthcare institutions. Three types of scheduling processes are common, each having its strengths and weaknesses (Burke et al., 2004):

- *Centralized Scheduling.* With a centralized scheduling practice, a departmental unit is responsible for creating schedules for its wards. The advantage is that a dedicated scheduling workforce can work specifically to ensure that demand is met in all the wards, increase cooperation between wards and exploit synergies. It also reduces the number of administrative tasks done by the medical workforce. A disadvantage is that manually creating such schedules in large departments might be challenging, and the workforce has less influence on working times.
- *Unit Scheduling.* Unit Scheduling refers to schedule practices where smaller units, such as a hospital ward, are responsible for the scheduling. Unit scheduling makes the communication distance between the schedulers and the employees smaller, making it easier to satisfy personal requests. Conversely, this

might lead to an unfair weighting of individual requests as the scheduler's relationship with the individual employees can influence their decisions.

- *Self-scheduling.* In self-scheduling, each employee constructs their own schedule, which a head employee adjusts to cover the demand. This enables a higher degree of autonomy and job satisfaction among the employees. On the other hand, it can be inefficient, as skilled employees use more of their time on planning. Large modifications might also be needed to meet the demand.

A common trait of the aforementioned scheduling processes is that they are time-consuming tasks, and the main difference lies in the level of the organization at which scheduling tasks are performed. Automating the schedule creation process thus constitutes a great potential in health institutions to reduce time spent on administrative tasks. It might free time for skilled nurses and doctors to improve quality and increase capacity for health services. A survey from Danish hospitals from 2008 shows that hospitals use, on average, 24 minutes on rostering per week per nurse (Range, 2021). With a rough estimate, a hospital employing 3000 nurses adds up to 62400 yearly working hours, equivalent to around 11 million DKK in yearly labor costs used for rostering. In light of this, it is evident that even a moderate reduction in time used for rostering can have a great impact.

2.4 Visma Resolve

The Visma Group, hereby referred to as Visma, is a Norwegian multinational company delivering software solutions and IT consulting services. It is considered one of the top 5 software companies in the EU and is the leading supplier of Cloud ERP systems in Europe. The company targets numerous markets and industries and is organized through 130 subsidiaries located at more than 150 locations in Europe and South America. *Visma Resolve* is Visma's optimization unit delivering commercial optimization software. Visma Resolve provides software solutions ranging from route-optimization for home care services, course scheduling at schools, and automatic kindergarten delegation in Norwegian municipalities to staff scheduling and project planning tools. Visma Resolve's focus is to develop algorithms that can be integrated into other software modules, which enables integration with complex ERP systems. Representatives from Visma Resolve believe that an increasing number of organizations have opened their eyes to the automation of complex planning tasks such as personnel rostering.

Automatic Rostering

Visma Resolve has developed an automatic rostering system to create fair and efficient schedules for the workforce that comply with laws and work regulations. The goal is for the system to be generalized and reusable across sectors with as little customization as possible. The system builds on the principles of centralized scheduling, but it has elements of self-scheduling by letting the employees include preferred working hours in the schedule generation. Their automatic rostering system utilizes modern cloud technologies to compute roster solutions. The algorithm

integrates with the Visma-owned *Medvind Workforce Management* system, which is a workforce management software available in Sweden. Several municipalities in Sweden have already put the system to use, and Visma reports a 60% reduction in schedule planning times among the active users.

Despite the success, Visma Resolve are continuously working on improving their solution approach. Their current system uses a heuristic algorithm to solve their rostering problem. However, they want to explore the possibility of incorporating machine learning techniques into their solution approach to improve performance and utilize the advantages of such technology.

2.5 Reinforcement Learning

This section is based on the introductory book on RL by Sutton and Barto (2018) and aims to provide the necessary background to understand the solution method we propose in Chapter 6.

RL is a branch of ML that concerns the mapping of states to actions in order to maximize cumulative rewards. The learner, commonly referred to as the *agent*, learns by exploring the environment through selecting actions and observing the state change and reward signal. RL can, for example, be applied in a game-playing setting where the ultimate goal is to learn a strategy that maximizes the probability of winning, or it can be used in robotics to learn how a mobile robot should interact with the environment.

2.5.1 Reinforcement Learning in the Machine Learning Domain

RL is not to be confused with the other two broad classes of ML, namely *supervised learning* and *unsupervised learning*. Supervised learning is defined by the use of a dataset in which each sample has several features and a label. The goal of supervised learning is to learn a model that generalizes its responses to inputs to give the correct label to samples not present in the dataset used for learning. The learning algorithm is supervised by an external supervisor during training, effectively telling the model how to respond to a subset of the data with the ultimate goal of training a model that generalizes on unseen cases. Two examples of supervised learning are classifying email as spam/no-spam and categorizing potential insurance customers into different risk groups using historical data. Although supervised learning is widely studied and suitable for many applications, it is not, in isolation, well suited for learning from interaction with an environment. In interactive problems, where the state space can be both unknown and stochastic, it is imperative that the agent can learn from its experience by interacting with the environment.

As the name suggests, unsupervised learning does not have an external supervisor, and the data used for training is thus not labeled. Although it is intuitive to think that RL can be categorized as unsupervised learning, the goals of these two learning

schemes are inherently different. Unsupervised learning is typically concerned with learning hidden structures in the unlabeled data, which could, for example, be clustering customers based on similar viewing patterns in order to make a recommender system. Another typical application of unsupervised learning is data exploration. On the other hand, the goal of RL is to maximize the cumulative reward, which is not addressed by unsupervised learning.

2.5.2 Elements of Reinforcement Learning

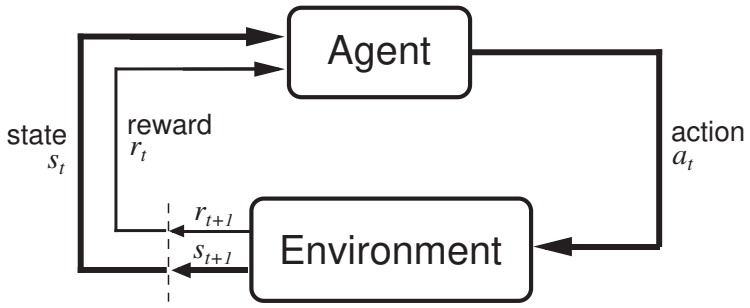


Figure 2.1: How an agent interacts with the environment in RL. The agent performs action a_t in state s_t to receive reward r_{t+1} and move to state s_{t+1}. Illustration from Sutton and Barto (2018).

Figure 2.1 depicts how an RL agent interacts with the environment; this relates to the four key elements of an RL system, namely a *policy*, a *reward signal*, a *value function*, and a *model of the environment*. Do note that the notation presented is only one of many ways to denote elements of RL.

The first key element of an RL system is a policy. A policy describes how the agent acts in the environment. Intuitively, given states as inputs, the policy describes which actions to take when in those states. π is often used as the symbol to describe a policy, and a policy can be either deterministic or stochastic. In a stochastic setting, $\pi(a|s, \boldsymbol{\theta})$ denotes the probability of selecting action a given a state s and a parameterization $\boldsymbol{\theta}$.

The goal of an RL problem is defined by the reward signal. A reward signal measures the agent's performance with respect to the goal and decides what is good or bad. For each action (i.e., step in the environment), the agent receives a numeric value from the environment called a reward. A reward at time step t is denoted r_t, and is often a stochastic result of choosing action a_{t-1} in state s_{t-1}, as Figure 2.1 illustrates. The only goal of the agent is thus to maximize the cumulative reward from interacting with the environment. In practice, the reward signals are often probabilistic functions with respect to the current state and the action taken.

As opposed to rewards, which give immediate feedback on whether or not a given state is good or bad, a value function describes the expected future value of being in a given state. For example, a state may often lead to a low immediate reward but

has a high expectancy of future rewards because the state is necessary to visit to get to some great series of states later on. The state-value function, which describes the value of being in a given state, is often denoted $V(s)$, and the action-value function, which describes the value of taking a given action in a given state, is often denoted $Q(s, a)$.

The last element of RL systems, which is optional, is a model of the environment. RL systems that make use of environment models use this to do planning, for example, to estimate future rewards and states. Methods that use environment models are called model-based methods, whereas methods that do not use models are called model-free and are explicitly trial-and-error agents.

2.5.3 Approximate Methods for Reinforcement Learning

In the simplest form of RL problems, the state and action spaces are so small that the value functions can be represented as arrays or tables. Solution methods for such problems are called tabular solution methods, and they can often find exact solutions. An exact solution, in this case, depicts a situation in which the optimal value function and policy are found. However, real-life problems seldom have such small state and action spaces, which is also the case for the RRP. This calls for so-called approximate methods, which find approximate solutions that can be applied adequately to problems of a much larger scale.

The goal of approximate methods is not to find the optimal policy and value function but to find good approximations even though the state space is too large to explore in its entirety. In order to do this, the approximations of the policy and value function based on limited interaction with the environment (i.e., state space) need to generalize well. Generalization from examples is what concerns supervised learning, as Section 2.5.1 discusses, so we can apply function approximation methods from this class of ML to parameterize the policy or value function. In relevance to the solution method presented in Chapter 6, we take a closer look at one domain of approximate methods for RL: deep RL.

Deep RL is the integration of ANNs and RL. The use of ANNs has been quite popular in the last several years, and this is due to the powerful function approximation and representation learning capabilities of ANNs (Arulkumaran et al., 2017, p. 1). Note that the terms ANN, Neural Network (NN), and Deep Neural Network (DNN) are often used interchangeably, although the latter is most commonly used to describe ANNs of a certain depth.

2.5.4 Artificial Neural Networks

The somewhat loose inspiration behind ANNs is the human nervous system. An ANN consists of several *neurons*, inspired by the neurons in the brain. The neurons are interconnected and can transmit signals, similar to the brain's synapses. Each connection, or edge, has a weight associated with it. A neuron can be considered

a processing unit with a summing part and an output part. The summing part computes a weighted sum over the neuron's inputs, and the output part produces a signal from this. The signal is referred to as the activation of the neuron (Yegnanarayana, 2009, Chapter 1). Figure 2.2 depicts the concept of a neuron in an ANN. The neuron in gray is connected to neurons x_1 and x_2 with associated weights w_1 and w_2, respectively. The first half of the gray neuron represents the summing part, and the second half represents the output part. f is called the *activation function*, and it takes the weighted sum as inputs and outputs some numeric activation value y. A common example of such an activation function is the logistic function, $f(x) = 1/(1 + e^{-x})$, where x is the weighted sum of the outputs from the previous layer. The bias is b, and it is also added to the weighted sum.

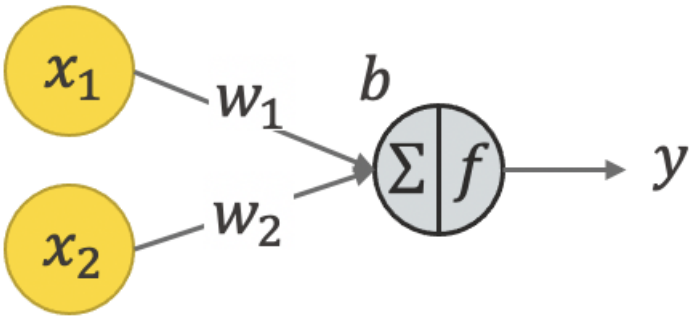


Figure 2.2: Example of a neuron with two incoming connections. Illustration from KNIME (2021).

Although there exists numerous architectural approaches to ANNs, we only elaborate on standard *feedforward ANNs*. A feedforward ANN is an ANN with one input *layer*, one output layer, and several intermediate hidden layers, where the latter decides the depth of the ANN. A layer in a feedforward ANN consists of many neurons, each of which is connected to all or some of the neurons in the next layer, except for the output layer, which has no subsequent layers. Figure 2.3 depicts a feedforward ANN with four layers: an input layer with four neurons, two hidden layers with four neurons each, and an output layer with two neurons. Effectively, this means that the input of the ANN is a vector of four numeric values, and the output is a vector of two numeric values.

An ANN is initialized to have small, random weights. In order to train an ANN, a function expressing its performance must be specified. Such a function is called a *loss function*, and can for example be the mean squared error:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} ||\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i||^2, \tag{2.1}$$

where $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are the target values and outputs of the ANN, respectively. n is the number of samples, and $\boldsymbol{y}_i$ and $\hat{\boldsymbol{y}}_i$ are target and output vectors for sample i, respectively. By computing the partial derivative of the loss with respect to any weight in the ANN, $\frac{\partial L}{\partial w_i}$, the *backpropagation algorithm* can use the chain rule

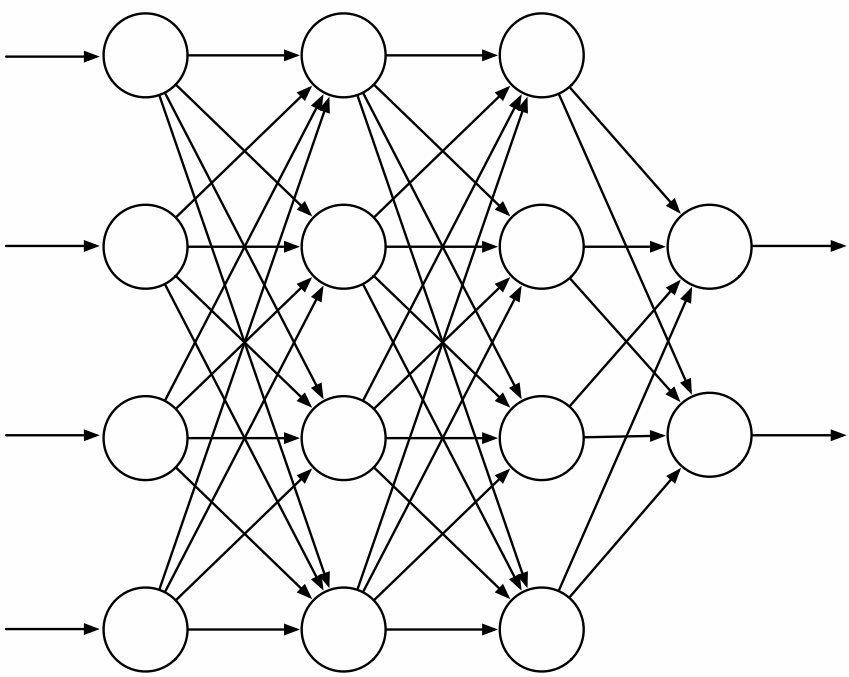


Figure 2.3: A standard feedforward ANN with a input layer, a output layer and two hidden layers. Illustration from Sutton and Barto (2018).

to propagate partial derivatives backward in the network and update the weights accordingly. This is what is happening when an ANN is being trained (i.e., learning the weights), and this is done over the entire training dataset multiple times. One such pass of sending all the training data through the network, propagating the partial derivatives backward, and updating the weights is called an *epoch*. How much the weights are updated according to the partial derivatives is called the *learning rate*. If the learning rate is zero, the updates will be zero, and the ANN is not learning anything. The entire process of updating the weights for several epochs is often called *training*.

The expressive power of an ANN is what makes it so suitable to use for function approximation in approximate methods for RL. In fact, an ANN with only a single hidden layer with a finite number of neurons with sigmoid activation functions can approximate any continuous function on a compact region of the ANNs input space to any degree of accuracy (Cybenko, 1989). Thus, an ANN is very suitable to parameterize the policy or value function. The weights of the edges in the network would, in that case, be the parameters of the policy or value function.

2.5.5 Proximal Policy Optimization

We have seen in Section 2.5.3 that the large state spaces and action spaces encountered in many RL problems make it infeasible to use exact methods and that we, therefore, make use of approximate methods. Furthermore, the expressive power of ANNs makes them suitable for function approximation in such methods. This section describes a state-of-the-art family of approximate RL algorithms named

Proximal Policy Optimization (PPO), first presented by Schulman et al. (2017). We focus on the actor-critic version of PPO to scope this section.

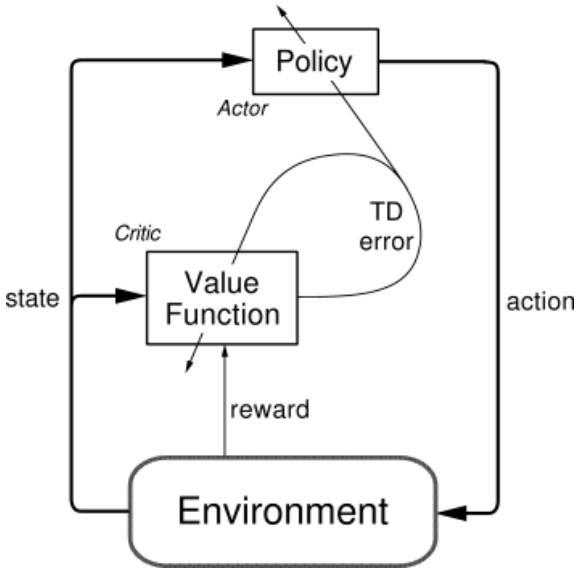


Figure 2.4: Actor-critic architecture. Illustration from Sutton and Barto (1998).

Figure 2.4 shows the actor-critic architecture for RL. In this architecture, the policy is also called the actor, and the value function is called the critic. In an approximate method for RL following the actor-critic architecture, both the value function and policy are approximated (i.e., learned). The value from the critic, or the value function, is used together with the rewards to update the actor. In other words, an agent consists of two parts: an actor and a critic.

Section 2.5.4 states that an ANN needs a loss function in order to compute gradients and update the weights of the network. PPO introduced a novel loss function for the actor:

$$L^{CLIP}(\boldsymbol{\theta}) = -\hat{\mathbb{E}}_t \left[\min(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right], \tag{2.2}$$

where $\boldsymbol{\theta}$ is the parameters of the policy, $\hat{A}_t$ is the advantage estimate, $r_t(\boldsymbol{\theta})$ is the ratio between the new and the old policy (i.e., before and after updating the weights), and ϵ is some small number. The ratio is given as

$$r_t(\boldsymbol{\theta}) = \frac{\pi(a_t|s_t, \boldsymbol{\theta})}{\pi(a_t|s_t, \boldsymbol{\theta_{old}})}, \tag{2.3}$$

and the clip function makes sure that the ratio is in the interval $[1-\epsilon, 1+\epsilon]$. Without going into too much detail, the advantage estimate, $\hat{A}_t$, is a measure of how good or bad an action was in a given state, and usually involves discounted rewards from subsequent timesteps and the value of being in the state from the critic. The intuition behind the L^{CLIP} loss is that we want to maximize (note the minus sign before the expression on the right size) the probability of choosing an action with a large advantage, but also to prevent the actor ANN from moving towards a policy where $r_t(\boldsymbol{\theta})$ is outside of the range $[1-\epsilon, 1+\epsilon]$. That is, we do not want the updates of the network weights to be too large.

Algorithm 1 PPO, Actor-Critic version

Input: N: number of iterations, T: maximum number of time steps, E: number of epochs

Output: $\boldsymbol{\theta}$: parameters of learned policy (actor), $\boldsymbol{\phi}$: parameters of learned value function (critic)

1: Initialize the parameters for the actor ANN, $\boldsymbol{\theta}$
2: $\boldsymbol{\theta}_{old} := \boldsymbol{\theta}$
3: Initialize the parameters for the critic ANN, $\boldsymbol{\phi}$
4: $\boldsymbol{\phi}_{old} := \boldsymbol{\phi}$
5: $n := 0$
6: **while** $n < N$ **do**
7: Select actions according to policy $\pi_{\boldsymbol{\theta}_{old}}$ in environment for T timesteps and receive rewards
8: Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ and discounted rewards $R_1, \ldots, R_T$
9: Train actor ANN ($\boldsymbol{\theta}$) and critic ANN ($\boldsymbol{\phi}$) for E epochs
10: $\boldsymbol{\theta}_{old} := \boldsymbol{\theta}$
11: $\boldsymbol{\phi}_{old} := \boldsymbol{\phi}$
12: $n := n + 1$
13: **end while**
14: **return** $\boldsymbol{\theta}$, $\boldsymbol{\phi}$

Algorithm 1 provides the pseudocode for the actor-critic version of PPO. Please note that this section has omitted to talk about the loss function of the critic ANN, but a typical loss function is the mean squared error as in Equation (2.1). In that case, $\hat{y}$ would be the value outputted from the critic ANN and y, the target, would be the discounted rewards. In Line 2 and Line 4 the old parameters for the actor and critic are initialized, respectively. This is so that the network can be trained through multiple epochs in Line 9 using stochastic gradient descent, which is also one of the main contributions of PPO. The old parameters are not updated until the network has trained for E epochs so that the old policy and value function remain constant and the L^{CLIP} loss function serves its purpose.

Chapter 3

Literature Review

This chapter discusses the extensive literature relevant to NNALNS and the RRP. Section 3.1 outlines the scope of our problem-related literature review, before Section 3.2 describes our procedure for finding relevant literature. Following, Section 3.3 presents the key aspects of PRPs and common modeling approaches in the literature. Lastly, Section 3.4 describes solution approaches that are relevant for this thesis, highlighting the use of ML in OR and PRPs.

3.1 Scoping the Resolve Rostering Problem

In OR, it is common to look at decisions and problems addressed in different organizational areas. Such areas can be financial, strategic, technological, or logistical. Labor is an organizational resource with limited capacity restricted both by quantity and labor regulations deciding the total amount of disposable hours. Individual employees are also distinguished by their competences and working positions, limiting how disposable work hours can be allocated. As the RRP directly regards planning and allocation of workforce resources, we consider it a resource capacity planning problem in the literature, according to the common control and planning framework proposed by Hans et al. (2012).

Another aspect to consider is the decision level within an organization where the RRP is applicable. Figure 3.1 depicts prior decision stages that highly affect the properties of an RRP instance. Firstly, an organization needs to decide the long-term demand for labor. Such decisions comprise recruitment decisions, where management decides on the size and competence mix of the workforce to cover future long-term demand. Workforce level decisions affect the RRP by setting an upper bound on available labor. With available labor established, the demand for specific competences in the planning period of the RRP needs to be forecasted. Traditionally, demand is determined manually by expert knowledge and experience. Finally, deciding how shifts are designed regarding start-time and duration needs to be addressed. The RRP presumes that the available workforce, the expected demand, and the possible shift types are known for the whole planning horizon. The RRP thus only consists of allocating shifts to the available workforce to meet the fore-

casted demand during the whole planning period. Based on these considerations, we position the RRP at an offline operational decision level within Hans et al. (2012)' framework.

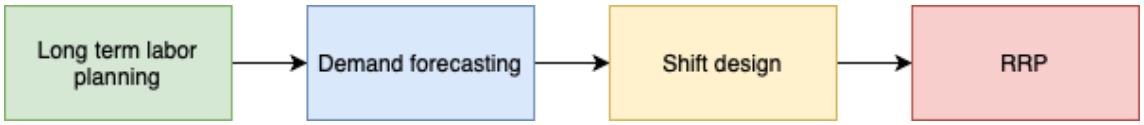


Figure 3.1: Prior stages to the RRP, affecting the properties of a problem instance.

3.2 Literature Search Strategy

This section presents our approach to finding relevant literature covering the key aspects of PRPs and different modeling and solution techniques. We search for scientific articles using the Google Scholar search engine, supplying keywords relevant to our problem positioning and focus areas. Our strategy to find relevant literature is to combine keywords covering the key aspects and focus areas of this thesis, both when it comes to the RRP and solving approaches in OR and ML. As the litterature on PRP and ML is quite extensive, we limit our search by examining the 10 most relevant results for each keyword, according to the search engine.

Table 3.1 lists the most interesting keyword combinations we have used. Some sources are found indirectly through references in literature from the search. This is especially the case when using extensive literature surveys such as Burke et al. (2004) for PRP literature, and Bengio et al. (2021) and Karimi-Mamaghan et al. (2022) for ML literature.

Table 3.1: Literature overview by search keywords.

Concept 1	**Concept 2**
Nurse/Personnel Rostering/Scheduling Problem	Fairness measures Literature review Survey State of the art Construction Heuristics Metaheuristics ALNS AI Machine Learning Reinforcement Learning
Combinatorial Problems	AI Machine Learning Reinforcement Learning
Operations research	AI Machine Learning Reinforcement Learning

3.3 Personnel Rostering Problems

This section presents the most important aspects of modeling PRPs and how previous works have approached them. These will provide a foundation for how we model the RRP.

3.3.1 Demand

Demand for labor is an important topic in PRPs. Demand coverage constraints ensure that the labor demand is met and can be considered a key identifier of a PRP. There are different approaches to modeling demand, and we use the taxonomy outlined by Ernst, Jiang, Krishnamoorthy and Sier (2004):

- *Task based demand* is to model demand as tasks with a duration, a time window, and skill requirements. The demand level is constant for any given task. Time windows may be longer than task duration, providing flexibility in when the tasks must be done (Smet et al., 2016). Alternatively, tasks will have a fixed time period, as in Smet and Vanden Berghe (2012); Beckmann and Klyve (2016). This is the norm for PRPs (Ernst, Jiang, Krishnamoorthy and Sier, 2004).
- *Flexible demand* rather separates the planning horizon into small, non-overlapping time intervals (time steps) that are each assigned a level or range of demand. Flexible demand is used by Grov et al. (2020), with a step length of 15-60 minutes. The RRP uses flexible demand with varying time steps.
- *Shift based demand* is the final approach to model demand, which assigns specific demand requirements to the defined shifts (Ceschia et al., 2019; Trilling et al., 2006).

Demand has a required level, denoting how many employees are needed. Kiefer (2015); Della Croce and Salassa (2014); Abobaker et al. (2011) define demand to only require a minimum number of employees and include demand as a hard constraint. In later years it has become common to model demand as soft constraints, with a minimum, desired, and maximum level, where deviation from the desired amount penalizes the schedule (Beckmann and Klyve, 2016; Grov et al., 2020; Turhan and Bilgen, 2020). Ceschia et al. (2019); Ouelhadj et al. (2012) use soft constraints but only penalize negative deviation from the desired level. The RRP has hard constraints on maximum and minimum demand and soft constraints on deviation from an ideal level within these bounds.

Finally, using demand with competence requirements is common. The competence requirements limit which employees can cover the demand, and they can either be defined as categorical or hierarchical (Van den Bergh et al., 2013). The former involves assigning a set of competences to each employee and demand, and any employee with one of the required competences can cover the demand. Ceschia et al. (2020, 2019); Mischek and Musliu (2016) use categorical demand. For hierarchical

competence requirements, the competences are ranked, and each nurse with a given competence can cover any demand that requires that or any lower-ranked competence (Beckmann and Klyve, 2016; Ramli et al., 2020). Abobaker et al. (2011) only include regular and senior nurses, where at least one senior nurse has to be assigned to each shift. The RRP uses categorical competences.

3.3.2 Workload Restricitons

Regulations often impose some constraints on the legality of shift patterns. One constraint that many papers use is limiting schedules to only assigning one shift per day per employee (Grov et al., 2020; Bai et al., 2010; Abobaker et al., 2011; Ceschia et al., 2019; Maenhout and Vanhoucke, 2010). Trilling et al. (2006) instead use an upper limit of twelve working hours per day. Some enforce a maximum number of accumulated hours per week (Beckmann and Klyve, 2016; Trilling et al., 2006), or a maximum number of shifts per week (Ramli et al., 2020; Abobaker et al., 2011). Finally, shift length can also be a subject of regulation, with minimum and maximum duration constraints (Grov et al., 2020). The RRP has the usual constraint of one shift per day as well as soft limits on the duration of shifts, in addition to the constraints in Section 3.3.3 on legality of shifts patterns according to Norwegian rest regulations

3.3.3 Rest

Another regulated aspect is employee rest. Daily rest is the most common type of rest constraint and is usually modeled as a hard constraint since many countries mandate a minimum amount of daily rest by law. One simple approach is a minimum rest period between shifts, used by Maenhout and Vanhoucke (2010), among others. Another common approach is that certain sequences of shift types are illegal, for example, that a morning shift cannot follow a night shift. Bunton et al. (2017); Ramli et al. (2020); Ceschia et al. (2019) incorporate this. Note that this requires predefined shift types to define illegal sequences or that this is done manually for instances. A final approach, used by Grov et al. (2020), is to require a consecutive period of rest within each day. The period could be within calendar days or 24-hour periods offset by a set time per employee. As this is a direct translation of Norwegian law (*Norweginan Working Environments Act §10-4*, 2015), the RRP incorporates this type of rest requirement.

Weekly rest is also usually required. Abobaker et al. (2011) sets a maximum number of working days per week, while Trilling et al. (2006) uses a limit on accumulated hours per week. This approach implicitly enforces a weekly rest amount and relies on other constraints, for example, consecutive days or partial weekends, to achieve more extended rest periods. The other approach is defining off shifts or off days and setting a lower limit of them per week. Grov et al. (2020) define a set of off shift with length upholding Norwegian law, which we emulate. Abobaker et al. (2011); Turhan and Bilgen (2020) rather require a specific number of off days per week, while Della Croce and Salassa (2014); Maenhout and Vanhoucke (2010) take this

further in requiring that these off days are consecutive. A rest-related issue that is rarely managed by solution approaches is breaks, which are usually assumed to be handled by employees on a day-to-day basis.

3.3.4 Contracted Hours

Employees often have a contractually determined number of hours to work each planning period. Grov et al. (2020) set the contracted hours as a hard ceiling on accumulated work hours and penalizes negative deviation. Another approach is to set upper and lower limits on deviation from contracted hours. Ramli et al. (2020) sets these limits as hard constraints. Ceschia et al. (2019); Abobaker et al. (2011); Trilling et al. (2006); Maenhout and Vanhoucke (2010) penalize, but allow, deviation outside these limits. Note that contracted and accumulated hours can also be represented as a number of shifts or days, dependent on shift structure. The RRP has an ideal number of hours per employee and penalizes deviation, without any upper or lower limit.

3.3.5 Weekends

It is agreed upon that working weekends is undesirable, but as demand is present during weekends, this is rather an issue of fair distribution. Bunton et al. (2017); Turhan and Bilgen (2020) set a hard maximum of working weekend days to distribute the allocation between employees, while Ceschia et al. (2019) define a maximum number of weekends an employee can work. Aside from simply working weekends, it is agreed upon that it is undesirable to work partial weekends. That is, working only on a Saturday or a Sunday. Grov et al. (2020); Kiefer (2015); Mischek and Musliu (2016) penalize these sequences, while Ceschia et al. (2019) determines these combinations to be illegal for select employees. The RRP penalizes a roster with partial weekends.

3.3.6 Consecutiveness

A final time-related aspect of shift pattern quality is an uniform distribution of work. Working isolated days is generally considered undesirable, and Turhan and Bilgen (2020); Guessoum et al. (2020) do not allow it, while Grov et al. (2020) penalize it. Another part of even work distribution is consecutive days. Working too few or too many consecutive days is undesirable. Many works specify this to penalize isolated days, and some extend this to isolated off days (Della Croce and Salassa, 2014). Ceschia et al. (2019); Ramli et al. (2020); Guessoum et al. (2020) set lower and upper limits on consecutive days. Della Croce and Salassa (2014) only use upper limits but disallow isolated days. Messelis et al. (2009) find that including consecutiveness tends to make problems harder to solve in experiments, but it is usually included in the literature nevertheless (Ouelhadj et al., 2012). The RRP has soft constraints that penalize both isolated working days, too many consecutive

days, and isolated off days.

3.3.7 Patterns

An aspect of schedule quality that is not strictly time-related but often combined with other constraints to enforce time off is shift-type patterns. Working many consecutive shifts of the same type can be considered undesirable. Della Croce and Salassa (2014); Turhan and Bilgen (2020); Guessoum et al. (2020) have an upper limit on consecutive shifts of the same type, while Abobaker et al. (2011); Mischek and Musliu (2016) apply this to night shifts. The RRP does not penalize a schedule where employees work many consecutive days with the same shift-type.

3.3.8 Preferences

Another aspect to consider are preferences requested by employees. Employees can influence the schedules by including preferences in rostering systems, leading to a higher degree of autonomy. Pato and Moz (2008) include preferences on a shift level where employees can record undesirable shifts, and assignments of such shifts are penalized in the objective function. Rönnberg et al. (2013) extend this by allowing both hard and soft preferences to provide flexibility in an automatic rostering system. A similar approach is taken by Grov et al. (2020), combining blocked days with a preference score for working a specific day. Beckmann and Klyve (2016) model preference by embedding personal inclinations for various shift patterns, e.g., their personal preference of working a set of consecutive days. The RRP considers both positive and negative preferences as soft constraints for working or not working during specific time intervals, as well as hard constraints on blocked hours throughout a day.

3.3.9 Fairness

In this thesis, we define fairness to denote the quality of a schedule for an individual employee in terms of the fairness aspects described in Sections 3.3.4-3.3.8. This definition is common in the literature, although some refer to balancing costs between employees as fairness. Including fairness in PRPs is considered a difficult task, as it requires quantification of subjective and qualitative evaluations. Furthermore, it is a situation-dependent parameter. Nevertheless, it is an essential aspect of the problem as research shows that "schedules perceived as unfair can lead to decreased job satisfaction, lower job performance and increased turnover rates" (Bard and Purnomo, 2005). In an overview of different PRP model designs by Burke et al. (2004), only a few extensively focus on the fairness aspect itself. However, Ouelhadj et al. (2012) argue that appropriate modeling of fairness is crucial for creating acceptance of schedules generated by automated scheduling systems.

A final aspect of fairness considers cost distribution between the employees. Wolbeck (2019) emphasize the following modeling techniques as the most common in the

literature to address fairness distribution:

1. Minimizing the sum of all individual costs for the employees.
2. Minimizing costs of the employee with the highest costs
3. Minimizing difference between the employee with the highest and lowest costs.
4. Minimizing the mean costs.
5. Minimizing the standard deviation between costs for the employees.
6. Maximizing the percentage of approved preference requests from the employees.

The various approaches have different characteristics associated with them and usually use some combination of these techniques. Grov et al. (2020), as well as the RRP, use a combination of (1) and (2).

3.3.10 Objective Function

All works on PRPs included in this literature review include fairness in their objective function. Aside from fairness, a majority of the works include demand deviation as a part of the objective function, (Grov et al., 2020; Ouelhadj et al., 2012; Ceschia et al., 2019; Kiefer, 2015; Mischek and Musliu, 2016; Turhan and Bilgen, 2020; Ceschia et al., 2020; Bunton et al., 2017) among others. Fewer include the highest-cost employee (Grov et al., 2020; Ouelhadj et al., 2012; Hadwan and Ayob, 2010). The RRP includes the highest-cost employee and demand deviation in the objective function in addition to the total fairness cost.

3.4 Solution Approaches

In the following section, we outline common approaches to solving PRPs. Section 3.4.1 focuses on exact methods, while Section 3.4.2 covers common heuristic approaches. Section 3.4.3 and Section 3.4.4 reviews how ML techniques are applied in the field of OR and personnel rostering.

3.4.1 Mathematical Programming

Mathematical programming is a straightforward approach to many optimization problems. Problems are usually modeled either as integer or mixed-integer formulations, which define a search space that must be enumerated to find the optimal solution (Lundgren et al., 2008). Mathematical programming techniques guarantee an optimal solution given enough computational time, as any candidate solution is either proven sub-optimal or optimal.

For PRPs, the complexity of evaluating the quality of rosters usually means that the search space is simply too large or of too high dimensionality to be effectively reduced. Most PRPs also belong to the class of NP-hard problems. Mathematical programming is therefore not dominant in nurse scheduling (Burke et al., 2006). Instead, using sophisticated heuristics is a common approach. This is mainly due to the individual preferences and the complexity of designing fair and balanced schedules (Thornton and Sattar, 1997).

3.4.2 Heuristics

Heuristics are proximal methods for solving computational problems, usually applied to achieve a time/quality trade-off. The definition of heuristics may vary, but we follow the definition by Wang (2010), which states that a heuristic is a methodology derived from reasoning that can be used to achieve solutions to computational problems through trial-and-error. Meta-heuristics are high-level heuristic frameworks that employ strategies in a local improvement procedure to escape local optima (Gendreau and Potvin, 2010).

Construction heuristics

A well-known class of heuristics is construction heuristics, aiming to construct solutions to a problem from an empty or partial solution. Within the academic field of personnel rostering, a limited amount of literature solely focuses on construction heuristics as the main research contribution. Van den Bergh et al. (2013) classify 14% of the papers they cover in their literature review to include a construction heuristic, while Ernst, Jiang, Krishnamoorthy, Owens and Sier (2004) identify around 19%. The more common use of construction heuristics in PRPs is to include them in earlier stages of a solution method or other sub-problems incorporated into the final solution by the main solution algorithm (Abobaker et al., 2011; Guessoum et al., 2020; Brucker et al., 2005). Forsyth and Wren (1997) use a construction heuristic on a PRP to evaluate different choices of parameters for their main algorithm by creating multiple solutions and calculating the probability of a solution being included in the main algorithm. Ross and Marfn-Blazquez (2005) follow a similar approach by using construction heuristics to attain metrics for the design of novel solution algorithms.

Improvement heuristics

Improvement heuristics aim to alter existing solutions to improve the objective value. They are often highly problem-specific as they often target specific aspects of a problem that might create an improvement. For PRPs, the most basic improvement heuristic is a single swap heuristic of shifts between two employees (Ceschia et al., 2020). Dowsland (1998) use chained neighborhoods defined by a sequential set of shift swaps for a fixed nurse, altering the whole shift pattern for a nurse, whereas

Bellanti et al. (2004) use a greedy local search procedure to complete partial solutions. Other improvement heuristics used on PRPs are swap-, change- and core shuffle-heuristics (Ceschia et al., 2020). A final type of heuristics is *destroy-repair heuristics*, referred to in this thesis as *destroy-repair operators*, which first destroys a solution by removing shift allocations and then repairs the solution by assigning new ones. These are typically used in ALNS (Ropke and Pisinger, 2006; Kiefer, 2015; Bilgin, 2012). NNALNS uses change- and swap operators and the standard destroy-repair operators.

Meta-heuristics

Meta-heuristics are problem-independent algorithmic frameworks aiming to decide the search procedure for iterative improvement of a solution. A popular category of meta-heuristics used for PRPs is Local Search-based meta-heuristics (Burke, De Causmaecker, Petrovic and Berghe, 2003). Simulated Annealing, Tabu Search, Large Neighborhood Search, and Adaptive Large Neighborhood Search are all prominent local search-based meta-heuristics. Another category is population-based meta-heuristics, where Genetics Algorithms are common for PRPs (Burke et al., 2004).

Simulated Annealing is traditionally used in PRP heuristics as the last improvement step in a larger heuristic scheme (Burke et al., 2004). Ceschia et al. (2020) use simulated annealing on NRP instances using a geometric cooling scheme. Hadwan and Ayob (2010); Turhan and Bilgen (2020) propose hybrid methods, where an improvement step using simulated annealing follows a preliminary construction step. The technique is also commonly applied as an acceptance criterion scheme in ALNS (Grov et al., 2020; Pisinger and Ropke, 2010; Kallestad, 2021).

Tabu Search keeps track of a tabu-list in the local search to prevent cycling around a local optimum. Ramli et al. (2020) use tabu search as their primary solution algorithm for solving an NRP with embedded nurse preference, only preceded by a semi-random initial step. Schrack et al. (2021) combine a tabu search with a genetic algorithm on an NRP with multiple preference options on shifts, days, and holidays.

Large Neighborhood Search (LNS) is a meta-heuristic framework proposed by Shaw (1998). It relies on a single pair of destroy and repair operators to construct a large neighborhood. Here the neighborhood of a solution is implicitly defined by the destroy and repair operators (Pisinger and Ropke, 2010). LNS was first successfully applied on the Vehicle Routing Problem (Shaw, 1998). Gregory (2010) use the LNS framework in an NRP setting with sliding time and nurse windows, restricting the destroy operator to focus on three random days or nurses at a time. Smet and Vanden Berghe (2016) apply LNS on large-scale shift assignment problems with multiple tasks, fixing a varying part of the solution and re-optimize it using mathematical programming to construct neighbors.

ALNS extends LNS by applying multiple competing destroy and repair operators. Each operator gets a probability weight which is updated during the search according to its performance. Ropke and Pisinger (2006) use the technique to solve vehicle routing problems with time windows. Kiefer (2015) apply ALNS on an NRP, pre-

calculating the probability weights of the operators using a simplified version of the algorithm. They design destroy operators that remove shifts based on relatedness measures between assignments and penalize assignments in the schedule. Their repair operators follow both a greedy approach and address the hard constraints of the problem. Bilgin (2012) use ALNS on a PRP, enforcing hard constraints on all operators. Furthermore, their operators use random permutations of shift assignments and single shift swaps between employees. Grov et al. (2020) use local search operators in addition to the destroy repair scheme in order to target specific aspects of the personnel schedule that affects the objective value.

Genetic Algorithms (GAs) are meta-heuristics, often described as population based, inspired by evolution, that evolves a population of possible solutions. Solutions evolve through filtering, mutation, and sharing of attributes to approximate an optimal solution (Dean, 2008). These have become popular for solving NRPs (Wu et al., 2013). Aickelin and Dowsland (2004) propose an indirect genetic algorithm for an NRP that encodes the search space based on permutations of nurses to handle the conflict between objectives and constraints, which they determine a common problem for GAs. Bai et al. (2010) creates a hybrid GA by combining a GA with simulated annealing to overcome the problem of evolving the solution at the later parts of the search.

Hyper-Heuristics (HHs) are types of high-level meta-heuristics that aim to intelligently choose, generate or combine lower-level heuristics to apply to a problem in a given situation (Burke, Kendall, Newall, Hart, Ross and Schulenburg, 2003). The term is, to some extent, used interchangeably with meta-heuristics in the literature, and the two frameworks are somewhat overlapping. However, the key difference, according to the definition by Burke, Kendall, Newall, Hart, Ross and Schulenburg (2003) is that HHs operate on a higher abstraction level than traditional meta-heuristics, searching in the heuristic space rather than in the solution space. HHs are an increasingly popular approach to solving Combinatorial Optimization Problems (COPs) (Burke, Kendall, Newall, Hart, Ross and Schulenburg, 2003). Burke et al. (2019) separate between two types of hyper-heuristics: *Selection HHs* and *Generation HHs*. Selection HHs aim to select among predefined low-level heuristics and Generation HHs generate new heuristics based on components from the low-level heuristics. Based on the definition of Selection HHs, we argue that ALNS can be viewed as a type of Selection HH due to its adaptive selection of operators. However, the ALNS framework directly evaluates what solutions to accept or not, putting it at the intersection between HHs and meta-heuristics. In this thesis, we focus on Selection HHs as it relates the most to our solution method presented in Chapter 6. HHs are identified as a promising field for integrating ML in OR, which we elaborate on in Section 3.4.3.

Burke, Kendall and Soubeiga (2003) use a Selection HH approach with a Tabu Search applied to timetabling and PRPs. Unlike in a textbook Tabu Search, the tabu list contains recently applied heuristics rather than solutions, thus preventing recent heuristics from being applied. Furthermore, heuristics are ranked according to their performance. Their algorithm produces comparable results to state-of-the-art problem-specific meta-heuristics. However, they report that it generalizes better to different problems. Anwar et al. (2014) apply a Harmony Search-based HH to an

NRP. Here sequences of heuristics are stored as vectors selected from a Harmony Memory Matrix (HMM). In the search procedure, the HMM is altered, creating new candidate heuristic vectors. The approach is similar to how genetic algorithms mutate solutions to create new ones. The heuristics used are simple moves and swaps of shifts in the schedule. Cowling et al. (2001) employs a hyper-heuristic to a scheduling problem, using a choice function to select low-level heuristics. The choice function includes information about the recent effectiveness of the heuristic, the recent effectiveness of consecutive pairs of heuristics, and the time since a heuristic was last applied. This approach yields better results compared to random and greedy selection of heuristics.

3.4.3 Machine Learning in Operations Research

In recent years, ML has gained growing attention and popularity within academic communities. It is considered one of the most rapidly growing research fields in computer science (Jordan and Mitchell, 2015). Although ML techniques have been around for decades, the internet has contributed to making enormous amounts of low-cost data available, giving ML methods increased applicability in several problem domains (Jordan and Mitchell, 2015). Furthermore, OR communities are increasingly looking to ML and ways of integrating ML in traditional OR methods, especially for solving COPs (Bengio et al., 2021). The possible synergies between ML and OR make ML a natural candidate for further research on optimization methods. Automating parts of developing algorithms to solve COPs has clear advantages, as the complex nature of the problems makes the design process time-consuming and requires a high degree of expert knowledge (Bengio et al., 2021).

Integration Architectures

Bengio et al. (2021) classify three different algorithmic architectures of how ML can be used in combination with OR methods.

1. *End-to-end learning.*
2. *Learn how to configure an OR method.*
3. *Integrate ML in OR algorithms.*

The term *end-to-end learning* is used to describe architectures where the ML models learn structures of a COP itself, aiming to create and improve solutions. Figure 3.2 illustrates end-to-end learning architectures. Gu and Hao (2018) use a pointer network with an ANN to solve the 0-1-Knapsack problem. A pointer network allows for encoding sequential input of varying sizes into an ANN and outputting a permutation of the input. In Gu and Hao (2018), the network trains on optimal samples of the problem in a supervised learning fashion, and the algorithm learns to obtain approximate solutions quickly. The disadvantage of using supervised learning to solve COPs is that it relies on having already optimal solutions as labels for training.

Bello et al. (2016) points out this issue and argues that the use of RL is more appropriate for solving COPs with ML. They apply a pointer network in an RL system to solve the Traveling Salesman Problem (TSP). Their algorithm solves the TSP with 200 nodes to near optimality without any heuristic design. However, Bello et al. (2016) report the method to be computationally expensive. Dai et al. (2017) uses a Graph Neural Network (GNN), a network architecture that can process graphs as input, to construct solutions to the TSP by iteratively adding nodes to the graph. GNNs differ from pointer networks as the output represents the value of adding a node n to a partially constructed graph instead of representing the graph itself. Dai et al. (2017) argues that the main advantage of their solution method is its generalizability across instances of different sizes; in addition, the method can be used on other COPs where solutions can be represented as a graph.

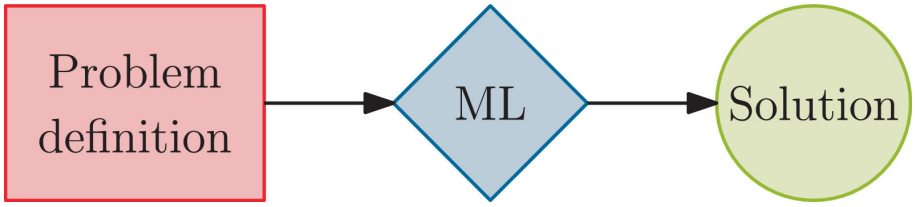


Figure 3.2: End-to-end learning. Figure from Bengio et al. (2021).

Configuring an OR algorithm is another application of ML in OR pointed out by Bengio et al. (2021). In such architectures, ML methods are used to affect the input parameters of an OR method, as Figure 3.3 shows. For example, in OR problems involving stochastic elements, ML methods can be used initially to predict the stochastic variables used in a deterministic model. A well-known example using this architecture is the Google Maps application, which finds the path with the shortest time distance between two locations (Derrow-Pinion et al., 2021). Here GNNs are used to accurately predict traffic conditions and estimate travel time in road segments. The information is passed to a Dijkstra shortest-path algorithm to minimize the estimated time of arrival (Derrow-Pinion et al., 2021; Mehta et al., 2019). Another approach to using ML to configure OR algorithms is to learn hyperparameters of the OR method. Kruber et al. (2017) use supervised learning to train an ANN that can decide which type of solver to apply to a MIP model. The purpose is to determine if a problem can be decomposed using Dantzig-Wolfe formulations and select a suitable solver to reduce solving time.

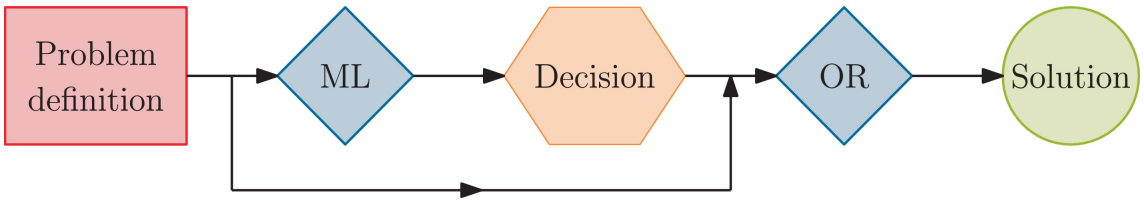


Figure 3.3: ML to configure OR. Figure from (Bengio et al., 2021).

The third architecture classified by Bengio et al. (2021) is the *integration of ML into OR algorithms*. For such architectures, OR algorithms interact with ML methods

during the solving process, as Figure 3.4 illustrates. NNALNS falls into this category, because it is a modification of ALNS that uses an ANN that parameterize the policy to select an operator in every iteration.

Alvarez et al. (2014) combine supervised ML with the Branch and Bound algorithm to learn branching strategies. The goal is to mimic a Strong Branching strategy by observing its branching decisions and learning these branching decisions at different stages of the search. As Strong Branching is a computationally expensive procedure, the rationale is that replacing it with a pre-trained neural network could yield a significantly faster search procedure with comparable performance. Other attempts at using ML methods to guide tree search algorithms are made by Balcan et al. (2018) and Sabharwal et al. (2012).

Using ML to assist meta-heuristics has also gained growing interest in recent years (Karimi-Mamaghan et al., 2022). ML is used to learn different aspects of meta-heuristics, like *algorithm selection*, *solution evaluation*, *evolution process* in genetic algorithms, *initialization* in improvement-based meta-heuristics and *operator selection* strategies (Karimi-Mamaghan et al., 2022). Nair et al. (2020) propose a Neural Large Neighborhood Search as an extension to the LNS framework proposed by Shaw (1998). The method aims to train a neural network to learn how to destroy a solution. They encode a MIP formulation into a Constraint-Variable Incidence Graph (CVIG) and use a Graph Convolution ANN to parameterize their policy and value function. Consequently, the ANN learns to select destroy variables in the solution. Subsequently, the solution is repaired using a MIP solver. Syed et al. (2019) also use the LNS framework combined with ML, but target the repair operator by using a trained ANN as an insertion operator. The ANN is trained using RL with an actor-critic architecture. The method is applied to a Vehicle Routing Problem with Time Windows and shows ability to scale well with larger problem instances.

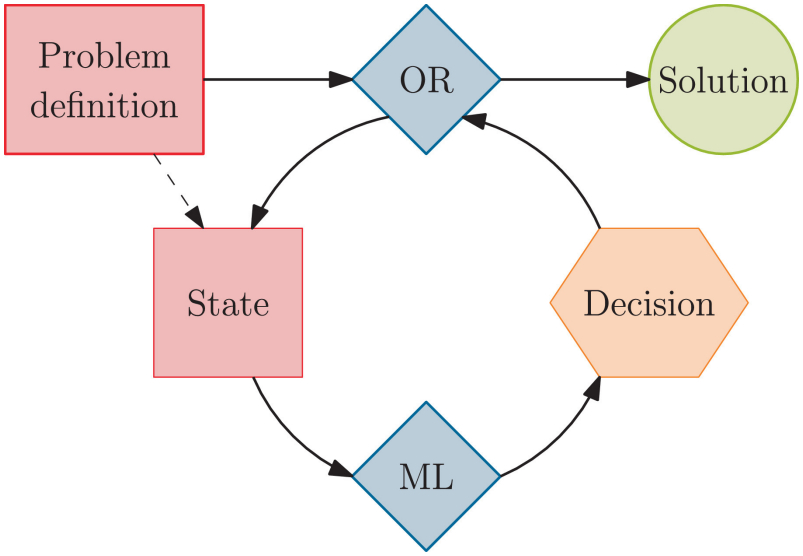


Figure 3.4: Using ML in OR algorithms. Figure from Bengio et al. (2021).

Operator selection is an application area for ML in meta-heuristics outlined by Bengio et al. (2021). The topic translates to Selection strategies in Selection Hyper-

Heuristics. Adaptive selection of operators is one of the core characteristics of the ALNS framework. However, the selection strategy in ALNS is stateless, thus only considering the values of the operator weights and not the state of the search process or solution. The motivation for introducing ML to learn intelligent operator selection strategies is rooted in the fact that different operators can be more effective at certain stages in the search. Hence, learning these patterns might give more efficient search trajectories.

Kallestad (2021) propose a Deep RL HH in their master's thesis. The algorithm uses an ANN trained offline with RL to select simple removal and insertion heuristics. They use search features to represent the state used as input to the network. Furthermore, the RL agent receives rewards according to the performance of the heuristics, with the same reward function scheme as in a traditional ALNS. The method shows promising results compared to an ALNS selection procedure. NNALNS also uses RL to train an ANN for operator selection, but with problem-specific operators and features in addition to the search-based features to solve the RRP.

Zhang et al. (2022) propose a similar Deep RL-based HH to solve COPs involving uncertainties. They use a Double Deep Q-network trained to select among a set of constructive heuristics to build solutions to a Container Terminal Truck Routing Problem with uncertain service times. The RL agent receives a reward according to how the low-level heuristic directly affects the objective function. Zhang and Lu (2008) use RL in an evolutionary algorithm to select mutation operators. Here the selection decision is what type of operator to choose at each mutation iteration of the evolutionary algorithm. At each stage, new operators are constructed through mutations of the selected operators. Zhang and Lu (2008) report that the algorithm performs the same or better than standard mutation strategies.

3.4.4 Machine Learning in Personnel Rostering Problems

In the literature, ML is applied to learn different aspects of PRPs. Kumar et al. (2019) use tensor operations and clustering techniques to learn constraints to be applied to an NRP. The rationale is that the mathematical modeling of an NRP is a time-consuming task that requires expert knowledge of mathematical modeling methods. In addition, schedule requirements vary between hospital wards. However, knowledge about the quality rosters is incorporated in manually created schedules as only the best schedules are included. Automatic learning by example schedules can help produce more precise models suited for specific wards. Their proposed algorithm is capable of capturing underlying characteristics of the example schedules and compares well with the actual constraints applied to the schedules.

Another way of using ML in PRPs found in the literature is to learn an evaluation of the solution quality. Václavík et al. (2016) argues that evaluating the roster quality in meta- and hyper-heuristics can be a computationally expensive task. They propose an ML method to accelerate the evaluation phase in heuristics. The method includes an ANN as a classifier that inputs two encoded solutions, one before and one after a change is made and outputs if the change would lead to an improvement or not. The algorithm includes a filtering mechanism to reject the majority of

potentially bad solutions.

Shi and Landa-Silva (2018) use an approximate dynamic programming approach to solve a multi-phased nurse rostering problem. Approximate dynamic programming strongly links to the term RL in the ML community, as both methods rely on the use of the Bellman Equation. Shi and Landa-Silva (2018) iteratively assign pre-generated weekly shift patterns to nurses and approximate a value function for several candidate allocations during a local phase procedure. Subsequently, the approximated value function is used in a global phase as a look-ahead policy. Langfeldt et al. (2021) use a similar approach by dividing the time horizon into weekly shift assignment problems and allocating pre-generated weekly shift patterns. The shift pattern generation phase uses k-means clustering to cluster similar shifts in order to reduce the number of possible shift pattern combinations.

Chen et al. (2021) use an ANN to guide a heuristic tree search to solve a PRP. The ANN inputs features of the tree search and statistical properties of the solution and output a value that reflects the distance a solution has to an optimal solution. This value is used to evaluate the solution produced by each change heuristic included in the algorithm and then choose which node in the tree to branch from. Thus, the algorithm relies on supervised learning based on pre-calculated optimal solutions to train the network. For training, pairs of features and the corresponding label are produced by applying the heuristics on the optimal solutions multiple times and assigning a value to the state, reflecting the number of times the heuristic is applied when the state is encountered.

Oberweger (2021) propose a method inspired by the Neural Large Neighborhood search presented in Section 3.4.3. The method is applied to a staff re-rostering problem, which is strongly related to PRPs. They use a GNN architecture to learn the destroy operator of LNS. For each day-nurse pair in the solution, the network outputs a probability of that pair being a part of a destroy set given the current state of the solution. After destroying the solution, it is repaired using a proposed MIP formulation. The method is reported to surpass the results of a LNS with a manually designed destroy operator on all measures, as well as being superior to a MIP formulation of the problem in terms of optimality gap.

3.4.5 Feature Engineering

Zheng and Casari (2018) defines feature engineering as the process of extracting numeric values from raw data and transforming them into formats that are suitable for ML. They also claim that the majority of time building ML models is spent on feature engineering and data cleaning. Feature engineering is a crucial step in an ML pipeline, as it provides the input to ML models, but is rarely discussed on its own (Zheng and Casari, 2018). Especially RL techniques rely on state representation. A challenge occurs when the dimensions of the states vary across instances. This section presents some approaches to feature engineering used in ML-assisted solution approaches to PRPs.

In most PRPs, a two-dimensional matrix of shift allocations can represent the state.

However, this requires that demand and competences are associated with shifts to calculate demand coverage. Most PRPs in the literature adhere to this, but in some works, like Grov et al. (2020), the demand is not given in terms of required shift allocations but rather for time intervals. These problem definitions require additional encodings to ascertain which employees cover what demand, expanding a potential raw feature vector. Regardless, such state representations can be used as input to an ANN. Chen et al. (2021) use the raw state representation as input to a supervised learning-scheme to predict the distance from candidate solutions to the optimal or near-optimal solution. They use this to assist the tree search in their solution method to a PRP. Beddoe et al. (2009) use a portion of the solution matrix around a soft constraint violation in a case-based reasoning approach to repair PRP solutions, which also share similarities with supervised learning (Mitchell, 1998). Gu and Hao (2018) use the graph representation of solutions as the input to their pointer network.

In exchange for a raw state representation, most proposed solution methods use a feature vector derived from solutions or search states as input to their ML-assisted solution approaches. However, which features to use is both varied and not a focus of most works, and while many works present their feature vectors, the impact of different vectors is rarely explored (Zheng and Casari, 2018). Messelis and De Causmaecker (2010) classify possible features into two categories; *static* and *dynamic*. As their name suggests, static features are constant attributes of problem instances. These can be simple statistics, like the number of employees or days, or more complex measures, like the average demand level compared to the number of available employees. Messelis and De Causmaecker (2010) collect a large set of static features, originally intended to determine the hardness of PRPs. Some works, like Alvarez et al. (2014), incorporate static features in their feature vector, often when learning across instances. The standard usage for static features is to describe the problem size and complexity, where complexity refers to a measure of how difficult it is to attain good solutions to a problem instance (Alvarez et al., 2014; Zarpellon et al., 2020).

In contrast, dynamic features describe aspects of the current solution beyond the static size and complexity. We choose to divide these into two categories: *search-based* and *problem-specific* features. Search-based features are aspects of the current state of the solution algorithm. These are natural inclusions in ANNs used to guide hyper-heuristics. Kallestad (2021) exclusively use the state of a neighborhood search as the input to an ANN to guide a HH. Works describing ANN-assisted branch-and-bound methods, such as Khalil et al. (2016); Alvarez et al. (2014), often use the status of the mathematical model of the current solution as features. Examples are statistics regarding sensitivity analysis of the mathematical model and the fixed variables. In this setting, Zarpellon et al. (2020) propose a feature vector that also includes features of the search tree, for example, depth and sparsity. To the best of our knowledge, most ML-assisted hyper-heuristics include search-based features in their feature vectors, indicating the power of such a state representation.

Problem-specific dynamic features are aspects of a solution interpreted in terms of the problem definition. In the case of personnel rostering, this can be statistics like shift assignments per day or demand coverage. This way of state representation

is less common than search-based features but still a common approach, especially when operator selection is the task of the ML model. Chen et al. (2022) use problem-specific features alongside search-based ones as input to a cross-instance ANN-policy for operator selection. Beddoe et al. (2009) use only problem-specific features in a case-based reasoning approach to nurse rostering. Compared to search-based features, problem-specific features are less commonly used. This discrepancy may be due to the reduced generalizability of using such features or how their distributions may be different across problem instances. Generalizability and stable distributions are both favored by the ML literature (Zheng and Casari, 2018). However, as one may construct operators with problem-specific aspects in mind, we argue that such features may be appropriate supplementation to search-based features in order to identify low-level strategies for improving solutions. NNALNS uses both problem-specific and search-based dynamic features as input for an ANN that parameterize the operator selection policy.

3.5 Our Contribution

In summary, this thesis presents a problem definition and solution method that draws inspiration from multiple similar works and contributes new ideas to solve the particular problem presented by Visma Resolve.

The RRP is a PRP and draws its main inspiration from Grov et al. (2020), as they also address a similar problem from Visma Resolve. The main difference from a regular PRP is the inclusion of rest rules in order to adhere to Norwegian labor law and the use of flexible demand to model the Visma dataset. Besides this, each aspect of the RRP is well represented among other works on PRPs.

In terms of feature engineering, we propose a novel feature vector that includes the regular search-based features found in most works on incorporating ML in solution methods for PRPs. However, we combine this with a new set of problem-specific features for the RRP into a vector of constant size; such a feature vector is not commonly used.

We present a NNALNS that can be considered an improvement-based selection hyper-heuristic, following the definitions of Burke, Kendall, Newall, Hart, Ross and Schulenburg (2003); Burke et al. (2019). The initial solution is constructed by solving a simplified MIP model, and NNALNS can be considered a modification of ALNS that uses a pre-trained ANN that parameterizes the operator selection policy. We combine ML and OR by using an ANN for operator selection, which is classified by Bengio et al. (2021) as incorporating ML into OR. This approach is inspired by Kallestad (2021). The ANN is trained by an implementation of a training algorithm from the state-of-the-art PPO-family of RL algorithms (Schulman et al., 2017).

Chapter 4

The Resolve Rostering Problem

This chapter presents the definition of the Resolve Rostering Problem, denoted the RRP. The problem definition is the same as in our specialization project (Langfeldt et al., 2021). Section 4.1 describes the constraints of the RRP, while 4.2 discuss demand modeling and Section 4.3 outlines the objective and related considerations. Finally, Section 4.4 discusses the assumptions made when formulating the RRP, and thus what is out of the scope of this thesis.

4.1 Constraints

A shift pattern is a sequence of shifts assigned to an employee. There are several constraints on the structure of these patterns. Section 4.1.1 outlines the hard constraints that determine the legality of a shift pattern, while 4.1.2 discusses what fairness aspects penalize or reward a schedule.

4.1.1 Hard Vonstraints on Shift Patterns

A shift pattern is a sequence of shifts that are assigned to an employee. There are numerous requirements that a shift pattern needs to meet to be considered a legal shift pattern. These requirements can be global in the sense that they need to be met for all patterns, or they can be employee-specific. For the sake of completeness and reproducibility, we elaborate on the requirements mentioned above.

Shared shift pattern requirements

A legal shift pattern can only contain one shift per day, and the day to which a shift belongs is predefined. Also, a shift cannot overlap with another shift regardless of the day the shift belongs to. This needs to be addressed in order to avoid overnight shifts overlapping with a shift the following day.

All legal shift patterns have requirements on the weekly minimum continuous stretch of rest. It can be helpful to view a continuous stretch of rest as a weekly off shift.

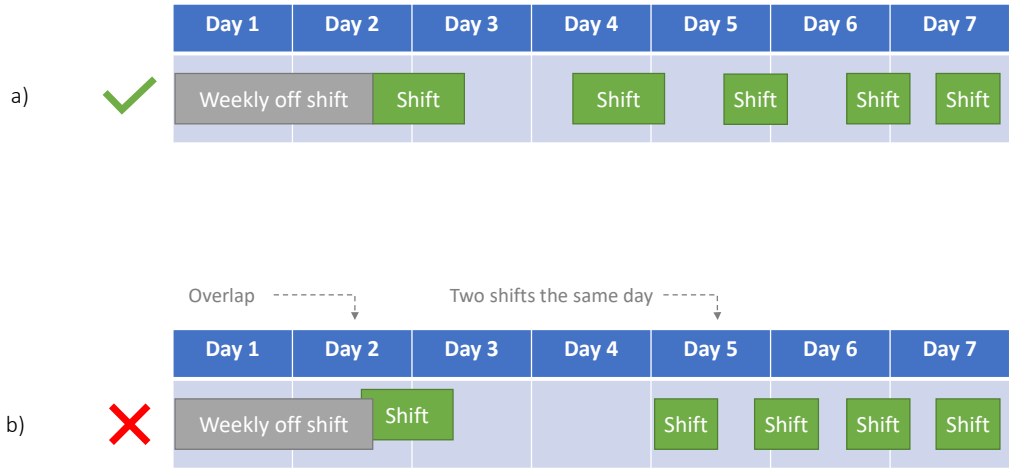


Figure 4.1: a) Example of legal shift pattern and b) example of illegal shift pattern.

Consequently, all legal shift patterns must contain one weekly off shift per week in the planning horizon, and the weekly off shifts cannot overlap with any work shifts in the pattern. Figure 4.1 illustrates an example of a legal and illegal shift pattern concerning the shared requirements.

Figure 4.1 a) illustrates a legal shift pattern with respect to the shared requirements. That is, the shift pattern contains no overlap, a maximum of one shift per day and one weekly off shift. Contrarily, Figure 4.1 b) illustrates an illegal shift pattern. The weekly off shift overlaps with a regular work shift on day two. In other words, the weekly requirement for minimum continuous rest is not met, although it appears to be due to the weekly off shift. Furthermore, if the overnight shift from day five to day six belongs to day five, the requirement of only having one shift per day is not met.

Employee-specific shift pattern requirements

Each employee has a list of blocked hours. This list states the time intervals when the employee cannot be assigned any shifts. A legal shift pattern for an employee cannot contain any shifts that violate this.

The daily rest rules state that for a defined 24 hour period each day, an employee should have a minimum amount of continuous rest. As the minimum amount of continuous rest and the definition of a 24 hour (i.e., offset) can vary among employees, this is considered a employee-specific requirement. To exemplify, the daily rest rule for a specific employee could be that between 06:00 one day and 06:00 the following day, the employee needs to have at least 9 hours of continuous rest. By looking at a specific shift and employee, there are three ways in which the daily rest rule can be violated:

1. The shift violates the daily rest rule single-handedly.
2. The shift violates the daily rest rule if it is combined with another shift.

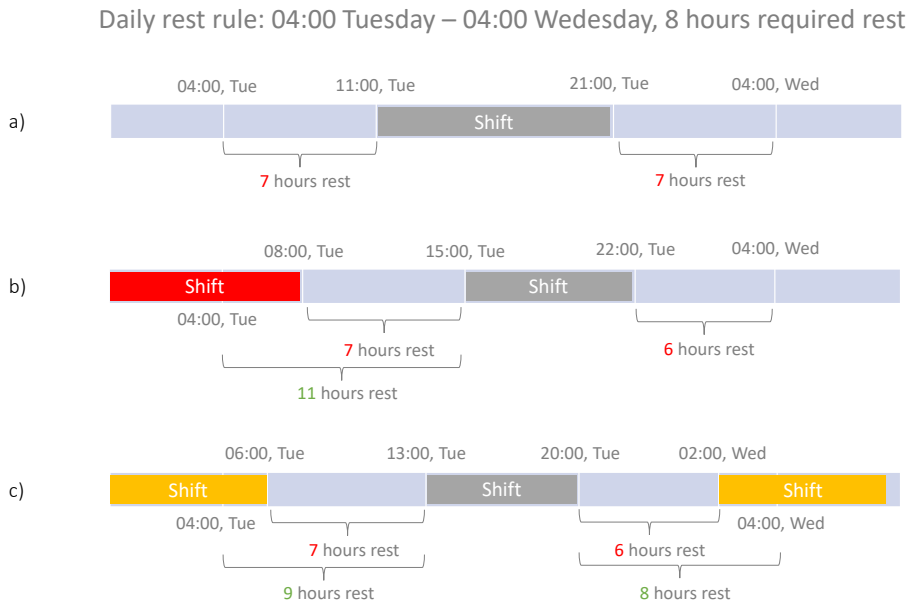


Figure 4.2: a) Example of shift that single-handedly violates the daily rest rule, b) example of shift that violates the daily rest rule if combined with another shift, and c) example of shift that violates the daily rest rule if combined with two other shifts.

3. The shift violates the daily rest rule if it is combined with two other shifts.

The three cases are illustrated in Figure 4.2.

The daily rest rule in Figure 4.2 states that between 04:00 and 04:00 the following day, there should at least be 8 hours of continuous rest. Figure 4.2 a) illustrates a shift that cannot be a part of any shift pattern for an employee with the corresponding day rest rule. Figure 4.2 b), on the other hand, shows that the shift indicated by the grey box can be a part of a legal shift pattern, but not when combined with the shift indicated by the red box. Lastly, Figure 4.2 c) shows that the shift indicated by the grey box combined with any one of the two shift patterns indicated by yellow boxes can be a part of a legal shift pattern. However, the grey shift cannot be combined with both of the yellow shifts.

The weekly and daily rest requirements are not standard to PRPs, but rather our modeling of the Norwegian Working Environments Act (*Norweginan Working Environments Act §10-4*, 2015). We made this adaptation to model Visma Resolve's specific scheduling problem.

4.1.2 Fairness Costs

Fairness refers to the quality of a schedule for the assigned employee. Qualities include collective preferences, like not working partial weekends, isolated days, or individual preferences for specific hours. The selection of these fairness constraints stems from the typical considerations in the literature discussed in Section 3.3.

First of all, there is a set of time-related fairness aspects. One of these is *contracted hours*. Working hours accumulate throughout the planning horizon, and deviation from the contractually set amount is penalized. Another time-related fairness aspect is *shift duration*, as there is a preferred duration for shifts, and shifts deviating from this range will penalize the schedule. We will not enforce hard limits on shift duration as problem instances determine the possible shifts. Finally, it is considered good to have longer weekly rest periods, and schedules with longer rest periods than the minimum duration will be rewarded, up to an upper limit.

Secondly, there is a set of global preferences that all schedules are penalized for breaking, regardless of the employee. It is desired to either work both days of a weekend or neither. Working one day of the weekend is called a *partial weekend* and penalizes a schedule. It is agreed upon that it is undesirable to work *isolated days*. They refer to working one day and not the previous or following day. Similarly, working *isolated days off* will also penalize the schedule. To work more than a predefined number of *consecutive days* is also considered undesirable.

Finally, individual preferences affect the fairness of a schedule. Preferences can denote an individual's wishes to work or not work at specific time intervals. They consist of the relevant time period and a score indicating the importance of the preference, which can be positive or negative, to distinguish favorable or unfavorable preferences. They can also model holidays by assigning the same negative preference spanning the holiday to each employee and similarly model agreed-upon preferences like working nights.

These fairness costs are accumulated per employee to calculate their *fairness costs*, which penalizes the schedule. Note that meeting preferences and scheduling longer weekly off shifts are considered desirable, reducing the fairness cost. A final preference is that employees are treated equally. Therefore, a schedule is also penalized for the worst individual fairness cost, which incentivizes scheduling employees to equal quality.

4.2 Demand

Demand denotes that some employees with specific competences are required to take a shift for the duration of the demand period. It spans a time period that may or may not be within a single day. A certain number of employees are required to take a shift that spans this period. To cover demand, an employee must have the corresponding competence. Competences are categorical, meaning that there is no hierarchical structure, and an employee must possess the actual competence required. The number of employees is given as the desired amount, referred to as *ideal demand*, but does also have a minimum and maximum legal limit. If the demand is satisfied within these limits, the schedule is considered feasible concerning demand. However, deviation from ideal demand penalizes a schedule.

4.3 Objective

The objective is to generate schedules that minimize aggregated fairness costs for all employees while minimizing the total deviation from ideal demand coverage. In addition to minimizing fairness for all employees, it is an objective to distribute fairness between employees evenly.

4.4 Assumptions

We have made five assumptions in the RRP. The planning period is isolated, meaning no information is carried into or out of the period. Some works include parameters that contain information carried on from earlier periods to enforce legality constraints on the first shifts (Ouelhadj et al., 2012). Secondly, there are no holidays in the problem. Each day is considered more or less the same, except for weekends and the individual preferences of the employees. Holidays can be incorporated into a problem instance by assigning these periods undesirable for all employees, similarly to what is done by Turhan and Bilgen (2020).

Finally, there are a few minor assumptions that are noted by other works. We assume that undesired and blocked hours have the same time-interval structures as shifts, rather than spanning more abstract periods, for example, "afternoons" or "one afternoon each week." Breaks are assumed to be covered by employees during shifts and will not be considered in the schedules. Finally, any overlap between shifts in order to transfer information is not required. Information transfer is not a standard inclusion in rostering problems but common practice at hospitals (Nurmi et al., 2016).

Several vital decisions or tasks are not in the scope of this thesis. The problem of meeting the long-term demand of labor is not within the scope of this thesis. Thus, the number of employees, their contracted hours, and their competences are given. Note that this means we do not consider wages and thus do not evaluate schedules on labor costs, except that schedules are penalized for over-staffing. Demand forecasting is also out of scope, as the problem instances determine desired demand and limits on deviation from ideal coverage. Hence, demand modeling is also out of scope. Moreover, this thesis does not cover shift design, as problem instances define the available shifts. All these decisions are vital to the resulting performance of the organization. However, as reducing costs is not the objective of the RRP, it is natural that these decisions are out of scope.

Aside from the aforementioned out-of-scope decisions determined by the problem instances, a few elements are not considered by this thesis. Uncertainty is a notable consideration. The desired demand is exact, and no forecasting nor stochastic optimization techniques are required to determine parameter values. Similarly, employee-specific blocked or undesired hours are all decided before the scheduling, so the availability of each employee is known. Hence, the schedules are not evaluated on their robustness nor flexibility but instead penalized for scheduling more or fewer employees than necessary. As we discuss in Chapter 3, we consider the RRP to be

offline operational and do not consider events that may occur during the planning horizon.

Chapter 5

Mathematical Model

This chapter describes the MIP formulation of the RRP we describe in Chapter 4. We define sets by upper case calligraphic letters, parameters by upper case Latin letters, and decision variables and indices by lower case Latin letters. We derive this model from the formulation of Grov et al. (2020). For sets, parameters, and decision variables, subscripts indicate indices, and we use superscripts to indicate particular meanings or specifications for sets.

5.1 Indices

The following indices are used in the model to denote different elements of sets.

e : Employee e
c : Competence c.
t : Time interval t.
s : Shift s.
r : Off shift r.
i : Day i.
j : Week j.

5.2 Sets

Below we list all sets in the model. Some sets are trivial, and not explained, but others are elaborated upon if relevant.

$\mathcal{I}$: Set of days.
$\mathcal{I}^{SAT}$: Set of Saturdays, excluding the final day in the planning horizon.
$\mathcal{J}$: Set of weeks.
$\mathcal{E}$: Set of all employees.
$\mathcal{E}_c^C$: Set of all employees with competence c.
$\mathcal{C}$: Set of competences.
$\mathcal{C}_e^E$: Set of the competences of employee e.
$\mathcal{T}$: Set of time intervals with demand.
$\mathcal{T}_r^R$: Set of time intervals overlapping weekly off shift r.
$\mathcal{S}$: Set of shifts.
$\mathcal{S}_i^I$: Set of shifts on day i.
$\mathcal{S}_t^O$: Set of shifts overlapping time interval t.
$\mathcal{S}_e^{VAL}$: Set of valid shifts for employee e, $\mathcal{S}_e^{VAL} \subseteq \mathcal{S}$.
$\mathcal{S}_{es}^{DRC}$: Set of shifts that breaks the daily rest rule for employee e if taken together with shift s, $\mathcal{S}_{es}^{DRC} \subseteq \mathcal{S}_e^{VAL}$.
$\mathcal{S}_{es}^{DRS}$: Set of shifts that breaks the daily rest rule for employee e if employee e is assigned to shift s and two of the shifts in $\mathcal{S}_{es}^{DRS}$, $\mathcal{S}_{es}^{DRS} \subseteq \mathcal{S}_e^{VAL}$.
$\mathcal{R}$: Set of weekly off shifts.
$\mathcal{R}_j^J$: Set of off shifts in week j.

$\mathcal{I}^{SAT}$ is the subset of $\mathcal{I}$ that contains Saturdays, excluding the last if the final day in the planning horizon is a Saturday, as this would not be considered a partial weekend. Demand in the RRP is modeled as time intervals t with demand, making up the set $\mathcal{T}$. Following the definition in Section 4.1, each shift in $\mathcal{S}$ belongs to the day it begins, and $\mathcal{S}_i^I$ is the subset of shifts that start on the day i. $\mathcal{S}_t^O$ is the set of shifts that overlap time interval t, and that can be assigned to employees to cover the demand in t.

A lot of constraints regarding rest and shift validity are handled by the sets $\mathcal{S}_e^{VAL}$, $\mathcal{S}_{es}^{DRC}$ and $\mathcal{S}_{es}^{DRS}$. $\mathcal{S}_e^{VAL}$ is the subset of $\mathcal{S}$ which one can assign to employee e. Shifts can be invalid by having such a long duration that it breaks daily rest rules outright or if they overlap with the employee's blocked hours, and these shifts are not added to $\mathcal{S}_e^{VAL}$. $\mathcal{S}_{es}^{DRC}$ is the set of shifts that cannot be assigned to employee e if that employee is already assigned to shift s without breaking the daily rest rule. $\mathcal{S}_{es}^{DRS}$ is the set of shifts that would break the daily rest rule if employee e is assigned to shift s and two shifts in $\mathcal{S}_{es}^{DRS}$, but where assigning s and one in $\mathcal{S}_{es}^{DRS}$ is legal. Daily rest rules are elaborated in Section 4.1.

While rest, both daily and weekly, is defined as the absence of working shifts, we choose to model weekly breaks as taking a *weekly off shift*. A weekly off shift restricts an employee from covering demand or taking other shifts when assigned to that weekly off shift and represents a longer continuous rest period. $\mathcal{R}$ is the set of all possible weekly off shifts. Each weekly off shift has a duration above the minimum required weekly rest, and it cannot span multiple weeks. $\mathcal{T}_r^R$ is the set of time intervals that overlap weekly off shift r, used to ensure that an employee is not covering demand while taking his or her weekly off shift.

5.3 Parameters

The following parameters and weights are used in our model of the RRP:
Instance Parameters

$\underline{D}_{tc}$: Minimum demand coverage for competence c in time interval t.
D_{tc} : Ideal demand coverage for competence c in time interval t.
$\overline{D}_{tc}$: Maximum demand coverage for competence c in time interval t.
V_r^R : Duration of off shift r.
$\underline{V}^R$: Minimum weekly off shift duration.
$\overline{V}^R$: Maximum rewarded weekly off shift duration.
B_e : Weekly contracted hours for employee e.
L^{CD} : Desired maximum number of consecutive days.
P_{es} : Preference score for employee e working shift s.
V_s^S : Length of shift s.
$\overline{V}^S$: Desired shift length, upper limit.
$\underline{V}^S$: Desired shift length, lower limit.
V_t^T : Duration of time interval t.

Demand is given as a upper-, ideal-, and lower level of coverage per time interval with demand t, per competence c, as the parameters $\overline{D}_{tc}$, D_{tc} and $\underline{D}_{tc}$ respectively. Off shift duration, V_r^R, as well as their minimum duration and maximum reward, $\underline{V}^R$ and $\overline{V}^R$ are given in hours. Contracted hours, B_e, denotes the contractually set number of work hours each week for employee e. L^{CD} determines the number of consecutive days an employee can work without incurring a penalty, and employees are penalized per day over this limit. We calculate preferences as a preference score, P_{es}, that is rewarded if employee e works shift s. A positive value indicates a desire to work that shift and thus reduces the fairness cost, while a negative value denotes a desire not to work and therefore increases the fairness cost. Duration of shifts and off shifts, V_s^S and V_t^T respectively, are given in hours, as well as the soft bounds on shift length, $\overline{V}^S$ and $\underline{V}^S$.

Weights

A^F : Adjustment factor for weighting the lowest individual fairness cost.
A_e^B : Adjustment factor for weighting cost of deviating from contracted hours.
W^F : Weight of the lowest individual fairness cost.
W^{D+} : Weight of excess demand coverage.
W^{D-} : Weight of deficit demand coverage.
W^R : Weight of weekly off shift score.
W^B : Weight of contracted hours difference.
W^{PW} : Weight of partial weekends.
W^{IW} : Weight of isolated work days.
W^{IO} : Weight of isolated off days.
W^{CD} : Weight of consecutive days.
W^P : Weight of preference score.
W^{SL} : Weight of shift duration.

We introduce two adjustment factors to dynamically value different parts of the objective function based on relevant instance attributes. The adjustment factor A^F scales the importance of the worst fairness cost, scaling with the number of employees. A_e^B scales the cost of deviating from contracted hours per employee based on their number of contracted hours. That is, a one-hour deviation for an employee with 13 hours of contractually set weekly work should be penalized harder than a one-hour deviation for an employee with 37 hours. The weights define the impact of breaking each soft constraint.

5.4 Decision variables

$$x_{es} = \begin{cases} 1, & \text{if employee } e \text{ is assigned to shift } s \\ 0, & \text{otherwise} \end{cases}$$

$$y_{etc} = \begin{cases} 1, & \text{if employee } e \text{ covers one unit of demand in time interval } t \\ & \text{with competence } c. \\ 0, & \text{otherwise} \end{cases}$$

$$w_{er} = \begin{cases} 1, & \text{if employee } e \text{ takes the weekly off shift } r \\ 0, & \text{otherwise} \end{cases}$$

$$\gamma_{ei} = \begin{cases} 1, & \text{if employee } e \text{ takes a work shift on day } i \\ 0, & \text{otherwise} \end{cases}$$

$$\rho_{ei}^{SAT}, \rho_{ei}^{SUN} = \begin{cases} 1, & \text{if employee } e \text{ works a partial weekend by working} \\ & \text{Saturday or Sunday, respectively} \\ 0, & \text{otherwise} \end{cases}$$

$$\sigma_{ei} = \begin{cases} 1, & \text{if employee } e \text{ works an isolated day on day } i \\ 0, & \text{otherwise} \end{cases}$$

$$\phi_{ei} = \begin{cases} 1, & \text{if employee } e \text{ takes an isolated off day on day } i \\ 0, & \text{otherwise} \end{cases}$$

$$\pi_{ei} = \begin{cases} 1, & \text{if employee } e \text{ starts a sequence of consecutive working days} \\ & \text{exceeding the desired limit on day } i \\ 0, & \text{otherwise} \end{cases}$$

$\delta_{tc}^+, \delta_{tc}^-$ = respectively excess and deficit demand coverage with respect to ideal. demand in time interval t for competence c.

f_e = variable for storing fairness-related costs for employee e.

g = variable for storing the highest fairness cost.

λ_e^+, λ_e^- = deviation between worked and contracted hours for employee e.

μ_{tc} = difference between covered and minimum demand for competence c in time interval t.

The variables x_{es}, y_{etc} and w_{er} can be considered the main decision variables, as these decisions determine a schedule. The other variables can be considered auxiliary variables, as they hold violations of soft constraints or other values used to calculate the objective value.

5.5 Objective function

$$\min z = \sum_{e \in \mathcal{E}} f_e \tag{5.1a}$$

$$+ W^F A^F g \tag{5.1b}$$

$$\sum_{t \in \mathcal{T}} V_t^T \sum_{c \in \mathcal{C}} (W^{D+} \delta_{tc}^+ + W^{D-} \delta_{tc}^-) \tag{5.1c}$$

The objective is to minimize the total fairness cost of the schedule and total deviation from ideal demand coverage. The first term (5.1a) is the total fairness cost of all employees, calculated based on how well the schedule adheres to the various fairness aspects. The second term (5.1b) adds the highest individual fairness cost to incentivize balancing the fairness costs between employees. The final term (5.1c) is the total deviation from ideal demand coverage. While the schedule must meet minimum demand coverage and not exceed the maximum, it will be penalized for deviating from ideal demand coverage. The weights, W^{D+} and W^{D-}, make it possible to weigh excess and deficit demand coverage differently. The deviation for each time interval t is scaled according to its duration, V_t^T.

5.6 Constraints

Demand constraints

$$\sum_{e \in \mathcal{E}_c^C} y_{etc} = \underline{D}_{tc} + \mu_{tc} \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{5.2}$$

$$\mu_{tc} \leq \overline{D}_{tc} - \underline{D}_{tc} \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{5.3}$$

$$\mu_{tc} + \underline{D}_{tc} - D_{tc} = \delta_{tc}^+ - \delta_{tc}^- \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{5.4}$$

The demand constraints ensure that demand-related variables are of the correct value. As we mention in Section 4.2, demand is given as legal bounds and an ideal coverage per time interval t and competence c. Constraints (5.2) enforce that μ_{tc}, the auxiliary variable storing demand coverage above the minimum amount is correct and, together with its non-negativity constraint, enforces that the schedule meets minimum demand. Constraints (5.3) enforce that demand coverage is not above maximum limits. Constraints (5.4) ensure that the demand deviation variables have correct values.

Work allocation

$$\sum_{s \in \mathcal{S}_i^I} x_{es} = \gamma_{ei} \qquad \forall e \in \mathcal{E}, i \in \mathcal{I} \tag{5.5}$$

$$\sum_{s \in \mathcal{S}_t^O} x_{es} = \sum_{c \in \mathcal{C}} y_{etc} \qquad \forall e \in \mathcal{E}, t \in \mathcal{T} \tag{5.6}$$

$$\sum_{c \in \mathcal{C}} y_{etc} \leq 1 \qquad \forall e \in \mathcal{E}, t \in \mathcal{T} \tag{5.7}$$

The work allocation constraints handle logical dependencies between variables necessary for theRRP. Constraints (5.5) and the binary restrictions on γ_{ei} enforce that at most one shift is taken each day per employee, as well as a correct value of γ_{ei}. Constraints (5.6) ensure that an employee is assigned to a shift when covering demand. Finally, while an employee can cover different competences throughout a day, Constraints (5.7) restrict employees from covering more than one unit of demand at the same time.

Rest constraints

$$2x_{es} + \sum_{s' \in S_{es}^{DRC}} x_{es'} \leq 2 \qquad \forall e \in \mathcal{E}, s \in \mathcal{S} \tag{5.8}$$

$$x_{es} + \sum_{s' \in S_{es}^{DRS}} x_{es'} \leq 2 \qquad \forall e \in \mathcal{E}, s \in \mathcal{S} \tag{5.9}$$

$$\sum_{r \in R_j^J} w_{er} = 1 \qquad \forall e \in \mathcal{E}, j \in \mathcal{J} \tag{5.10}$$

$$|\mathcal{T}_r^R| w_{er} \leq \sum_{t \in \mathcal{T}_r^R} (1 - \sum_{c \in C} y_{etc}) \qquad \forall e \in \mathcal{E}, r \in \mathcal{R} \tag{5.11}$$

As elaborated in Section 4.1.1, an employee must have a continuous period of rest of a certain length every individually defined 24-hour period. Shifts can be long enough for this to be impossible and are thus not added to $\mathcal{S}_e^{VAL}$. Beyond these shifts, different sequences of shift allocations can break this daily rest rule. Some pairs of shifts span periods where, if taken by the same employee on subsequent days, they will break the rest rule. For each shift, we, therefore, add all other shifts where this would be the case to the set $\mathcal{S}_{es}^{DRC}$. This set is created per employee, as they can have differently defined 24-hour periods. The set will contain shifts on days on either side of the relevant day that are too close to the relevant shift. With this set, Constraints (5.8) ensure that these illegal pairs of shifts are not taken. The final way shift sequences can break the daily rest rule is if an illegal sequence of three shifts is taken. Consider three shifts belonging to subsequent days. The shift we are evaluating belongs to the day in the middle. Either shift can be taken together with the shift we are considering, as the employee will have satisfactory rest. However, if an employee takes a shift on both days as well as the middle day, they will not provide satisfactory rest. All shifts with this property, evaluated based on the shift s, are added to the set $\mathcal{S}_{es}^{DRS}$. With the set $\mathcal{S}_{es}^{DRS}$, Constraints (5.9) ensure that the daily rest rule is not broken in this way. The set is, of course, only a subset of $\mathcal{S}_s^{VAL}$,

as many sequences of shifts can be assigned on consecutive days without breaking the daily rest rule.

Constraints (5.10) enforce that each employee takes exactly one weekly off shift each week, thus enforcing the weekly rest rule, while Constraints (5.11) enforce that employees do not cover demand when assigned to a weekly off shift.

Fairness

Fairness costs can be considered the main bulk of the objective function. Constrains (5.12) calculate individual employees' fairness costs. Terms (5.12a)-(5.12d) calculate the fairness cost for isolated working days, isolated off days, consecutive days, and partial weekends, respectively. Terms (5.12e)-(5.12g) calculate how much each assigned shift's duration deviates outside desired limits, the deviation from contracted hours, and preference score, respectively. Term (5.12h) calculates the duration of each weekly off shift above the minimum amount and will not give rewards for durations longer than the maximum rewarded weekly off shift duration, $\overline{V}^R$. Lastly, Constraints (5.13) make sure that the variable g is equal to the fairness cost of the employee with the highest fairness cost.

$$
\begin{aligned}
f_e = {} & W^{IW} \sum_{i \in I} \sigma_i && && \text{(5.12a)} \\
& + W_{IO} \sum_{i \in I} \phi_{ei} && && \text{(5.12b)} \\
& + W_{CD} \sum_{i \in I} \pi_{ei} && && \text{(5.12c)} \\
& + W_{PW} \sum_{i \in I^{SAT}} (\rho_{ei}^{SAT} + \rho_{e(i+1)}^{SUN}) && && \text{(5.12d)} \\
& + W_{SL} \sum_{s \in S_e^{VAL}} \max\left(V_s^S - \overline{V}^S, \underline{V}^S - V_s^S, 0\right) x_{es} && && \text{(5.12e)} \\
& + W_B A_e^B (\lambda_e^+ + \lambda_e^-) && && \text{(5.12f)} \\
& - W_P \sum_{s \in S_e^{VAL}} P_{es} x_{es} && && \text{(5.12g)} \\
& - W_R \sum_{r \in R} \min(V_r^R - \underline{V}^R, \overline{V}^R) w_{er} && \forall e \in \mathcal{E} && \text{(5.12h)}
\end{aligned}
$$

$$
g \geq f_e \qquad \forall e \in \mathcal{E} \qquad \text{(5.13)}
$$

Constraints (5.14)-(5.18) ensure that the fairness-related auxiliary variables have the correct values. Constraints (5.14) ensure that deviation from contracted hours is correct, while Constraints (5.15) enforce that partial weekends are penalized. Note that a single Saturday or Sunday at the edges of the planning horizon is not counted as a partial weekend, and $\mathcal{I}^{SAT}$ does not include a Saturday if it is the last day of the planning horizon. Constraints (5.16)-(5.17) ensure that isolated workdays and off days are counted correctly, while Constraints (5.18) count the number of days in a sequence of consecutive days above the desired limit.

$$\sum_{s \in \mathcal{S}^{\mathcal{VAL}}} V_s^S x_{es} + \lambda_e^+ - \lambda_e^- = |\mathcal{J}| B_e \quad \forall e \in \mathcal{E} \tag{5.14}$$

$$\gamma_{ei} - \gamma_{e(i+1)} = \rho_{ei}^{SAT} - \rho_{e(i+1)}^{SUN} \quad \forall e \in \mathcal{E}, i \in \mathcal{I}^{\mathcal{SAT}} \tag{5.15}$$

$$\gamma_{ei} - \gamma_{e(i-1)} - \gamma_{e(i+1)} \leq \sigma_{ei} \quad \forall e \in \mathcal{E}, i \in \{2, 3, ..., |\mathcal{I}| - 1\} \tag{5.16}$$

$$\gamma_{e(i-1)} - \gamma_{ei} + \gamma_{e(i+1)} - 1 \leq \phi_{ei} \quad \forall e \in \mathcal{E}, i \in \{2, 3, ..., |\mathcal{I}| - 1\} \tag{5.17}$$

$$\sum_{i'=i}^{i+L^{CD}} \gamma_{ei'} - L^{CD} \leq \pi_{ei} \quad \forall e \in \mathcal{E}, i \in \{1, 2, ..., |\mathcal{I}| - L^{CD}\} \tag{5.18}$$

Variable definition constraints

$$x_{es} \in \{0, 1\} \qquad \forall e \in \mathcal{E}, s \in \mathcal{S}_e^{VAL} \tag{5.19}$$

$$y_{etc} \in \{0, 1\} \qquad \forall e \in \mathcal{E}, t \in \mathcal{T}, c \in \mathcal{C}_e \tag{5.20}$$

$$w_{er} \in \{0, 1\} \qquad \forall e \in \mathcal{E}, r \in \mathcal{R} \tag{5.21}$$

$$\delta_{tc}^+, \delta_{tc}^- \in \mathbb{Z}^+ \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{5.22}$$

$$\rho_{ei}^{SAT}, \rho_{e(i+1)}^{SUN} \in \{0, 1\} \qquad \forall e \in \mathcal{E}, i \in \mathcal{I}^{\mathcal{SAT}} \tag{5.23}$$

$$\gamma_{ei}, \sigma_{ei}, \phi_{ei}, \pi_{ei} \in \{0, 1\} \qquad \forall e \in \mathcal{E}, i \in \mathcal{I} \tag{5.24}$$

$$\mu_{tc} \in \mathbb{Z}^+ \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{5.25}$$

$$\lambda_e^+, \lambda_e^- \ \geq 0 \qquad \forall e \in \mathcal{E} \tag{5.26}$$

$$f_e, \text{ free} \qquad \forall e \in \mathcal{E} \tag{5.27}$$

$$g, \text{ free} \tag{5.28}$$

Chapter 6

Solution Method

This chapter presents our solution method for the RRP. Firstly, Section 6.1 describes how we adapt ALNS for the RRP, which entails all of the operators, stopping criteria, acceptance criteria, reward function, how we construct the initial solution, and some adjustments concerning infeasible solutions. Following, Section 6.2 describes a modification to ALNS that we name NNALNS. Instead of using an adaptive weight scheme to alter the probability distribution for operator selection, NNALNS uses an ANN that is pre-trained using RL to map any given state to a probability distribution over the operators. In order for the state of the search to be used as input to an ANN, we perform feature engineering to encode the state as a feature vector of fixed size.

6.1 Adaptive Large Neighborhood Search

Based on Ropke and Pisinger (2006), this section describes ALNS for the RRP in detail to set the foundation for the NNALNS algorithm in Section 6.2. Firstly, Section 6.1.1 outlines the algorithmic framework for ALNS. Following, Section 6.1.2 describes how an initial solution is found, before Section 6.1.3 presents the operators designed for the RRP. Section 6.1.4 and Section 6.1.5 describes the different acceptance and stopping criteria, respectively. Lastly, Section 6.1.6 elaborates on the operator selection and adaptive weights, Section 6.1.7 describes the reward function, and Section 6.1.8 explains which measures are taken with respect to feasibility.

6.1.1 Algorithmic Framework

ALNS is an extension of the LNS framework, where several sub-heuristics, or operators, compete during the neighborhood search. Each operator has a weight associated with it, and the weights of the operators are updated according to a reward function, which Section 6.1.7 describes. At each operator decision point, an operator is sampled according to the probability distribution given by the weights. Figure 6.1 illustrates the flow of ALNS.

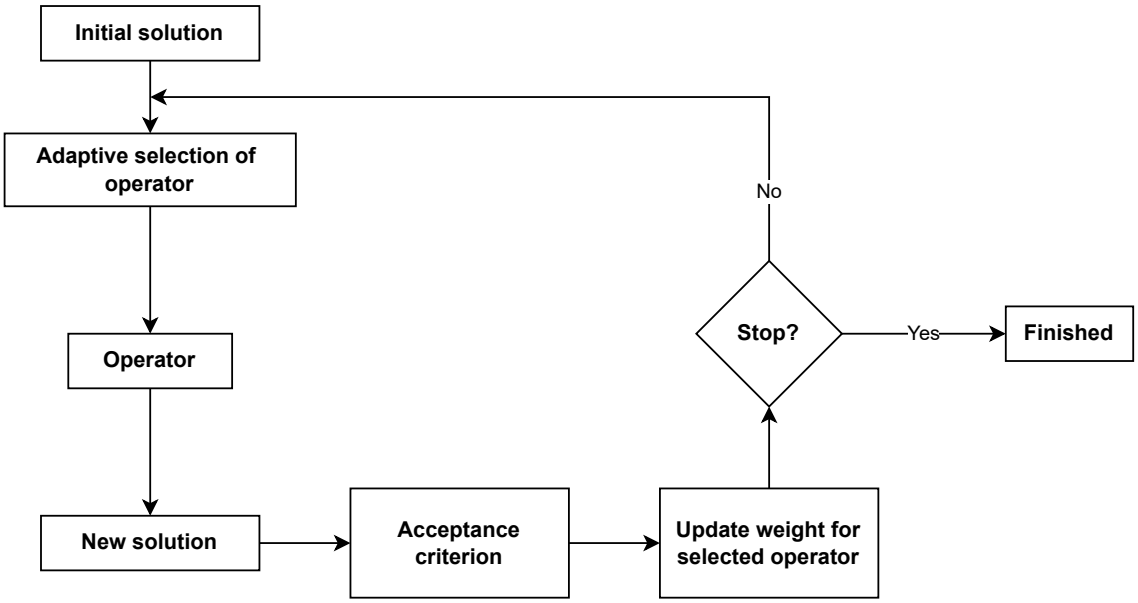


Figure 6.1: Algorithmic sketch of ALNS for the RRP.

Algorithm 2 provides the pseudocode for ALNS. Input to the algorithm is an initial solution from which to start the search, and the output is the best solution found throughout the search. In Line 1, the weights of the operators are initialized to the same value so that the probability distribution is uniform. The current solution, $x_{current}$, and the best solution, x_{best}, are initialized to the initial solution in Line 2-3. Line 4-14 constitute the main loop of ALNS. Firstly, an operator is selected according to the probability distribution given by the weights of the operators. An operator is commonly referred to as the pair of a destroy operator and a repair operator. However, as Section 6.1.3 describes, we also propose some hybrid operators which cannot easily be split into a destroy and repair operator. Following the operator selection, the selected operator is applied to the current solution, resulting in the candidate solution $x_{candidate}$. The candidate solution is either accepted or rejected according to some acceptance criterion. If the possibly new current solution is better than the current best solution, x_{best} is updated. Lastly, the weight for the selected operator is then updated in Line 13 according to how well the operator performed. This main loop is repeated until the stopping criterion is met and ALNS terminates with the best solution, x_{best}.

Algorithm 2 ALNS

Input: x_0: initial solution
Output: x_{best}: best solution found

```
1: InitializeWeights( )
2: x_current := x_0
3: x_best := x_0
4: while stopping criterion not met do
5:     o := SelectOperator()
6:     x_candidate := o(x_current)
7:     if Accept(x_candidate, x_current) then
8:         x_current := x_candidate
9:     end if
10:    if f(x_current) < f(x_best) then
11:        x_best := x_current
12:    end if
13:    UpdateWeights( )
14: end while
15: return x_best
```

6.1.2 Initial Solution

ALNS requires an initial solution as a starting point for the search. To construct such a solution for the RRP, we propose a simple construction model solved as a MIP model reflecting the mathematical model presented in Chapter 5, but with an altered objective function. The altered objective, shown in Equation (6.1), is to minimize the total number of shifts assigned in the solution.

$$\min z = \sum_{e \in \mathcal{E}} \sum_{s \in \mathcal{S}_e^{VAL}} x_{es} \tag{6.1}$$

The altered objective removes the soft constraints from consideration, focusing on creating a feasible solution to the RRP satisfying rest rules and minimum demand coverage. Consequentially, it expands the solution space making the problem significantly easier to solve and allowing us to obtain an initial solution to the RRP more quickly.

6.1.3 Operators

Operators are an integral part of the ALNS framework. The operators alter a current solution to produce a new candidate solution. Consequently, the set of possible operators to apply to a problem directly defines the search space in the neighborhood search. As operators are highly problem-specific, they need to be designed to reflect the problem to be solved. We propose several types of operators for the RRP that capture various properties of the problem. For inspiration, we use operator types that are commonly used for NRPs according to Pillay and Qu (2019), being larger destroy-repair operators, shift swap, and shift change operators.

Destroy Operators

The destroy operators in ALNS aim to destroy different parts of a solution to ensure sufficient exploration of the neighborhood. In the RRP, destroying a solution corresponds to removing shift allocations and the corresponding demand coverage from the solution. Selection of which shift allocations to remove is what differentiates the destroy operators for the RRP. Each destroy operator returns a destroyed solution along with a destroy-specific set, d, which describes which part of the solution that has been removed:

We categorize the destroy operators in this thesis into four types:

- *Week destroy*: Remove all shift allocations within k number of weeks in the solution.
- *Employee destroy*: Remove all shift allocations for k employees in the solution.
- *Day destroy*: Remove all shift allocations on a targeted set of days in the solution.
- *Shift destroy*: Remove a targeted set of shift allocations from the solution.

The different types reflect what part of the solution the operators target. However, a selection strategy is needed to choose the particular part to destroy. We use four selection strategies:

- *Greedy*: Select greedily according to the objective function.
- *Uniform random*: Select stochastically according to a uniform probability distribution.
- *Weighted random*: Select stochastically according to a weighted probability distribution.
- *Targeted*: Select specifically targeted elements.

Figure 6.2 shows how the various destroy types are combined with the selection strategies.

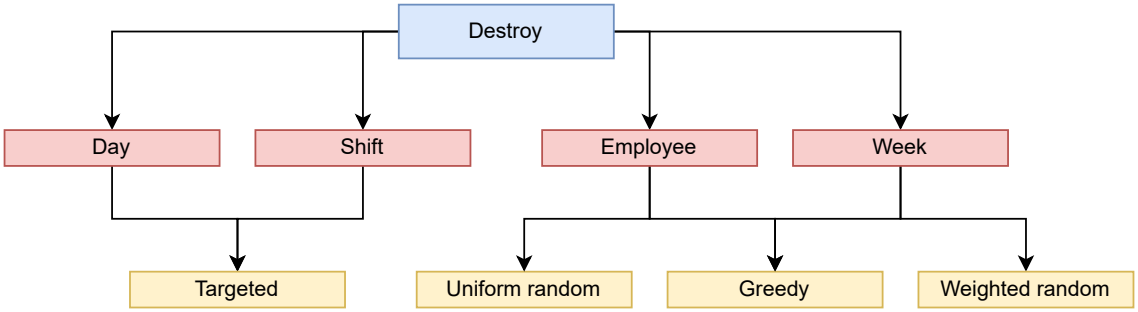


Figure 6.2: Destroy operator types and selection strategy combinations.

Week Destroy

Since many of the costs of an RRP solution are week specific or related to sequences of shifts, destroying a whole week is a natural choice. A week can be regarded as an isolated schedule allowing us to calculate the soft constraint violation costs associated with each week in isolation, thus measuring separate parts of the solution. To select which week to destroy, we use three selection strategies. The uniform random selection strategy selects random weeks of the solution. Contrary, the greedy selection strategy chooses the week with the highest costs. This is calculated according to the soft constraint violations, being the total fairness costs of the employees in (5.1a), the cost of the employee with the highest fairness costs in (5.1b), and demand costs in (5.1c), though measured within a weekly timeframe. Lastly, the weighted random selection strategy falls between the two, selecting weeks according to a weighted probability distribution based on the total costs for each week. The three selection strategies give us three levels of stochasticity, which we argue is beneficial as it diversifies the neighborhood search. The destroy-specific set for the week destroy operators contains the selected weeks.

Employee Destroy

Removing shift allocations for employees is an appropriate choice for the RRP as it directly addresses the fairness costs in the objective function (see Equation (5.1a)). We use the same three strategies as for the week destroy operators to select which employee to remove from the solution, namely uniform random selection, weighted random selection, and greedy selection. However, for the greedy and weighted selection strategies, the cost associated with each employee is calculated according to the individual fairness cost, defined by Equation (5.12). The destroy-specific set for the employee destroy operators contains the employees where shifts have been removed.

Day Destroy

We include day destroy operators to target specific days of the solution. Since most of the objective value costs are related to sequential patterns in the solution (e.g., consecutive days costs, isolated day costs), it is hard to measure the costs for a specific day. Thus, the aforementioned greedy, weighted random, and uniform random selection strategies do not apply to these operator types. Instead, we apply a targeted day selection strategy. The *Demand Day Destroy* operator selects all days in the solution where the minimum demand has not been met, thus targeting hard-constraint violations linked to demand coverage. The *Weekend Destroy* operator destroys whole weekends in the solution to allow a repair operator to restructure the shift allocation on weekends, targeting the partial weekend costs (Constraints (5.15) in the objective function. The destroy-specific set for the day destroy operators contains the destroyed days.

Shift Destroy

The shift destroy operators allow focused targeting of shift allocation removals within the solution. We include a *Partial Weekend Destroy* operator that removes all shift allocations that cause a partial weekend in the solution, thus explicitly targeting partial weekend shift allocations. In addition, the *Demand Shift Destroy* opera-

tor removes a given percentage of the shift allocations on days where demand is not covered, chosen stochastically according to a uniform probability distribution. This selection strategy is beneficial in situations where only small re-allocations are needed to satisfy demand on a given day. Instead of removing all shifts on that day, fewer shifts are removed, keeping the majority of the shift allocations.

Repair Operators

In the ALNS framework, repair operators are used to reconstruct destroyed solutions. The repair operators need to handle both cost- and feasibility-aspects of a problem, and it is beneficial to have repair operators that address different problem aspects to create a variation in how solutions are reconstructed. For the RRP, reconstruction means assigning shifts to employees to cover demand.

All the repair operators for the RRP have the same structure. However, they differ by using different combinations of selection strategies to choose shifts, employees, and competences to add to the solution. Algorithm 3 outlines the procedure for how our repair operators reconstruct destroyed solutions. Firstly, in Line 2, the algorithm constructs a set, S, that contains the candidate shifts extracted from the destroy-specific set, d. Such filtering narrows the possible shift allocations, forcing the repair operator to focus on areas that have been destroyed by a destroy operator. The repair operator then selects a shift from S in Line 4, creates a set, E, of candidate employees who can work the shift in Line 5, and selects the employee to assign the shift among the candidates in Line 10. Finally, it selects the competence the employee uses during the shift in Line 11. The chosen allocation is then added to the solution in Line 12. This procedure repeats until a stopping criterion is met.

Algorithm 3 Repair

Input: x: Destroyed Solution, d: Destroy-specific set
Output: x': Repaired Solution

1: $x' := x$
2: $S :=$ GetIncludedShifts(x', d)
3: **while** stopping critera not met **do**
4: $\quad s :=$ GetShift(x', S)
5: $\quad E :=$ GetPossibleEmployees(x', s)
6: $\quad$ **if** $|E| = 0$ **then**
7: $\quad\quad I := I \setminus s$
8: $\quad\quad$ **Continue**
9: $\quad$ **end if**
10: $\quad e :=$ GetEmployee(x', s, E)
11: $\quad c :=$ GetCompetence(x', s, e)
12: $\quad x' :=$ AssignShift(x', e, s, c)
13: **end while**
14: **return** x'

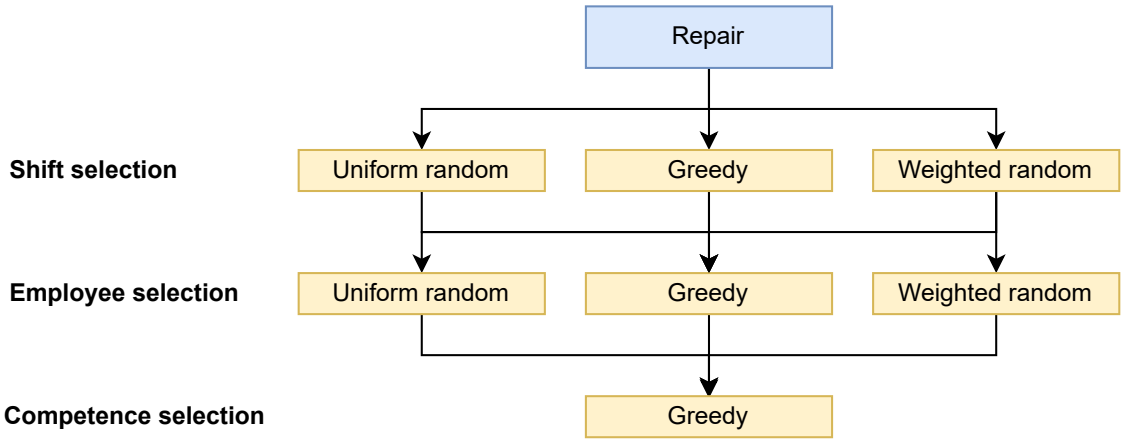


Figure 6.3: Selection strategy combinations for the repair operators.

The repair algorithm makes it possible to construct various repair operators that use different selection strategies for shifts, employees, and competences to reconstruct a solution. Figure 6.3 shows all the selection strategies we use to form RRP repair operators.

Shift Selection

The positive impact of adding a shift to a schedule, independent of which employee takes it, is linked to how much demand the shift potentially can cover. Hence, a natural approach is to select the shift that covers the most unsatisfied demand, as this facilitates covering demand and thus enforces feasibility. However, as excess demand coverage is penalized in the objective function in the RRP, it is preferable to select a shift that overlaps time intervals with the highest *demand deficit* from the ideal demand. This is calculated according to Equation (6.2), where $\mathcal{O}_s$ is the set of time intervals overlapped by shift s, D_{tc} is the ideal demand coverage and D^F_{tc} is the already filled demand for competence c in time interval t.

$$\arg\max_{s \in \mathcal{S}} \sum_{t \in \mathcal{O}_s} \sum_{c \in \mathcal{C}} (D_{tc} - D^F_{tc}) \tag{6.2}$$

We use three different shift selection strategies: uniform random, greedy, and weighted random selection. The uniform random strategy selects shifts among the shift included in S according to a uniform probability distribution. The greedy selection strategy selects shifts from S according to Equation (6.2). Furthermore, the weighted shift selection strategy selects shifts according to a weighted probability distribution based on the covered demand deficit associated with each shift.

Employee Selection

The employee selection process in the repair operators is twofold. Firstly, there are hard constraints to consider, such as daily and weekly rest rules that might not be fulfilled if the employee is assigned the selected shift. The repair operators filter such violations and construct a set E of candidate employees. Employees are selected from this set according to different strategies depending on the repair operator.

The employee selection strategies we use are structured similarly to many of the destroy operators, namely with three levels of stochasticity. To define a greedy selection choice, we measure the impact on the objective function if the shift is assigned to each candidate employee and choose the employee that gives the best impact on objective value. The weighted random selection strategy uses these values to calculate a weighted probability distribution and selects accordingly.

Competence Selection

As an employee can cover demand for different competences at different time intervals during the same shift, the repair operators greedily choose the competences based on the demand deviation in each time interval overlapped by the selected shift and the employee's competences.

Destroy-Repair Combinations

The destroy and repair operators combine to form a complete operator that outputs a new solution to the RRP. Appendix B.1 provides a complete list of all destroy-repair pairs used in this thesis.

Hybrid Operators

In addition to the classical destroy-repair operators, we also implement some operators that do not follow such a framework. These are either operators tailored to specific fairness aspects or operators intended to impose small changes to a solution.

Swap Operators

Swap operators constitute a class of operators that swap the shift allocations of pairs of employees, with the intent to impose small changes to a solution. Swapping shifts has the advantage of scheduling the same shifts after the swap. Demand coverage should therefore remain the same, as each employee covers what the other drops, except for employees with different competences. Algorithm 4 shows the simple swap operator, which selects two pairs of employees and shifts and swaps their shift allocations on these days.

Algorithm 4 Swap operators

Input: $x_{current}$: current solution
Output: $x_{candidate}$: candidate solution

for k iterations **do**
  e_1 := ChooseInitialEmployee($x_{current}$)
  d_1 := ChooseInitialDay($x_{current}, e_1$)
  e_2, d_2 := ChooseOtherDayAndEmployee($x_{current}, e_1, d_1$)
  $x_{candidate}$:= SwapShifts($(e_1, d_1), (e_2, d_2)$)
end for
return $x_{candidate}$

Employees are selected either uniformly random, greedily, or targeted. The greedy choice is to select the employees with the highest fairness cost. The targeted selection is to select one employee that has weeks without weekly off shifts, paired with a random employee. Days are selected either uniformly random or greedily, where the greedy choice is to select the day where unassigning the employee's shift would improve the objective value the most. Figure 6.4 illustrates the variations of swap operators.

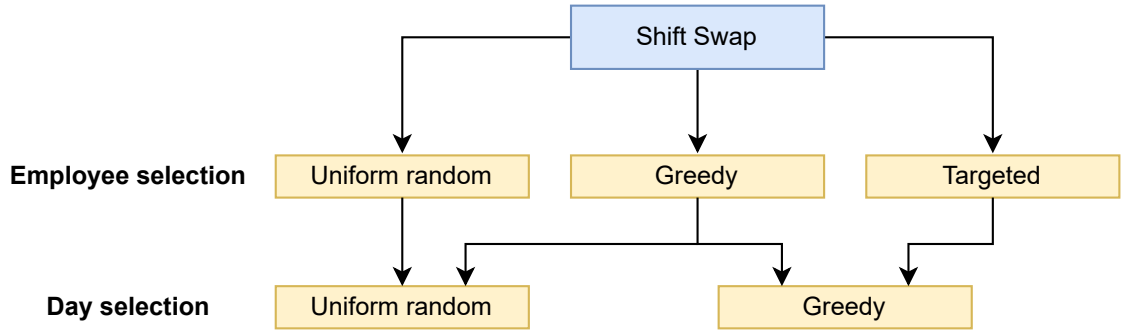


Figure 6.4: Selection strategy combinations for the shift swap operators.

Aside from selection strategies, swap operators can also differ in the number of swaps performed. We refer to this as the *size* of the swap operator.

Shift Change Operators

The other class of hybrid operators consists of the shift change operators. They change a schedule by assigning employees to a different shift, or no shift, on a specific day. These operators do not benefit from maintaining demand coverage, which the swap operators do. However, these operators can improve solutions where shift assignments are *ineffective*, that is, the shifts incur unnecessary fairness costs without covering demand. Algorithm 5 shows the basic shift change operator. A set of employee-day pairs are selected to be changed, and the main loop greedily selects the best shift to assign based on an estimation of the costs of these shift assignments.

Figure 6.5 shows the various selection strategies. We choose employees either randomly or based on individual fairness costs. However, as it is difficult to ascertain the fairness costs of a single day without costly estimations, the days to change are chosen randomly. The number of changes may vary. One tailored shift change operator is to select employees and days that incur isolated off day-penalties. We include this operator to reduce isolated off day- and partial weekend penalties and improve the score for weekly off shift duration (Constraints (5.12b), (5.12d) and (5.12h).

Algorithm 5 Shift Change Operators

Input: $x_{current}$: initial solution
Output: $x_{candidate}$: new solution

$pairs :=$ GetEmployeeDayPairs($x_{current}$)
$x_{candidate} := x_{current}$
for pair in pairs **do**
    e, $d :=$ employee, day from pair
    $impacts :=$ list of zeros with length equal to the number of shifts
    $shifts :=$ list of shifts valid for e on d
    for s in shifts **do**:
        $impacts[s] :=$ GetIndividualAssignmentImpact($pair, s$)
    end for
    $s_{best} :=$ argmax($impacts$)
    $x_{candidate} :=$ AssignShift($x_{candidate}, e, d, s_{best}$)
end for
return $x_{candidate}$

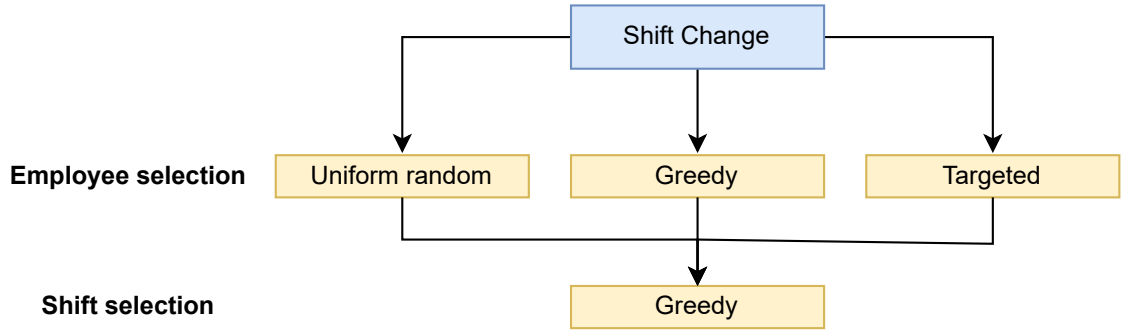


Figure 6.5: Selection strategy combinations for the shift change operators.

6.1.4 Acceptance criteria

After applying the selected operator on the current solution during an iteration of ALNS, a new candidate solution is found. The candidate solution can be accepted or rejected, dependent on the acceptance criterion and quality. We include three acceptance criteria in this thesis, which are not problem-specific and are widely studied in the literature (Pillay and Qu, 2019, p. 63).

Hill Climbing

Algorithm 6 provides the pseudocode for hill climbing. Only a candidate solution that improves the objective value is accepted using hill climbing. If ALNS is stuck in a local optimum, the search is over because there are no improving solutions in the neighborhood.

Algorithm 6 Hill Climbing

Input: $x_{current}$: current solution, $x_{candidate}$: candidate solution
Output: *accept*: boolean indicating if the candidate solution is accepted

1: $accept := false$
2: **if** $f(x_{candidate}) < f(x_{current})$ **then**
3: $\quad accept := True$
4: **end if**
5: **return** *accept*

Threshold Acceptance

The second acceptance criterion is threshold acceptance, which was first presented by Dueck and Scheuer (1990). Algorithm 7 provides the pseudocode for threshold acceptance in the case of a minimization problem, like the RRP. As for hill climbing, the input is a candidate solution and the current solution. Threshold acceptance accepts a solution if the candidate solution constitutes an improvement in objective value or if the absolute value of the ratio between the difference in objective value between the current and candidate solution and the objective value of the candidate solution is below a threshold. The threshold is updated at each iteration according to the equation

$$T_{t+1} = T_t - k,$$

where T_t and T_{t+1} are the thresholds before and after the update, respectively, and k is some constant. The threshold update does not need to be restricted to being linear.

Initially, the threshold is set to some value so that the acceptance criterion is not too restrictive; we want ALNS to explore the solution space in earlier iterations. As the threshold decreases, the acceptance criterion becomes more conservative.

Algorithm 7 Threshold Acceptance

Input: $x_{current}$: current solution, $x_{candidate}$: candidate solution
Output: *accept*: boolean indicating if the candidate solution is accepted

1: $T :=$ GetThreshold()
2: $\Delta := f(x_{current}) - f(x_{candidate})$
3: $accept := False$
4: **if** $diff > 0$ or Abs($\frac{\Delta}{f(x_{candidate})}$) $\leq T$ **then**
5: $\quad accept := True$
6: **end if**
7: UpdateThreshold()
8: **return** *accept*

Simulated Annealing

Kirkpatrick et al. (1983) first introduced simulated annealing, before Ropke and Pisinger (2006) applied it for the first time in the context of ALNS. The inspi-

ration behind simulated annealing is drawn from statistical mechanics; annealing in metallurgy involves heating and controlled cooling of a material. Similarly, the temperature in simulated annealing is initially high before it is cooled down in a controlled manner. The temperature directly affects whether a candidate solution is accepted, comparable to the threshold in threshold acceptance. However, simulated annealing is stochastic because the temperature affects the probability of selecting a non-improving solution.

Algorithm 8 constitutes the pseudocode for simulated annealing. An improving candidate solution is always accepted using simulated annealing as the stopping criterion. If the candidate solution is not an improving solution, the expression $e^{(f(x_{current})-f(x_{candidate}))/T}$, where T is the temperature, is compared to a random number p sampled uniformly from the continuous interval $[0, 1]$. The candidate solution is accepted if p is smaller than or equal to the above expression. If all other things are left constant, a decrease in the temperature will lower the probability of selecting a non-improving solution. At the end of each iteration, the temperature is updated. Although many variants exist, this thesis uses the update given by $T_{t+1} = \alpha T_t$, where α is some small number less than one.

To find the initial temperature, L number of iterations of ALNS where all candidate solutions are accepted is run. During this preliminary run, the difference, Δ, in Line 2 is recorded for all non-improving solutions. After L iterations, the initial temperature is calculated as

$$T_0 = -\frac{\overline{\Delta}}{\ln(0.8)},$$

where $\overline{\Delta}$ is the average difference. The rationale is that the initial temperature should be so that there is approximately an 80% chance of accepting the average non-improving solution.

Algorithm 8 Simulated Annealing

Input: $x_{current}$: current solution, $x_{candidate}$: candidate solution
Output: *accept*: boolean indicating if the candidate solution is accepted

1: $T :=$ GetTemperature()
2: $\Delta := f(x_{candidate}) - f(x_{current})$
3: $accept := false$
4: **if** $\Delta < 0$ **then**
5: $\quad accept := True$
6: **end if**
7: $p =$ RandomNumber() ▷ Random number in the interval $[0, 1\rangle$
8: **if** $\Delta \geq 0$ and $p \leq$ Exp$(-\frac{\Delta}{T})$ **then**
9: $\quad accept := True$
10: **end if**
11: UpdateTemperature()
12: **return** *accept*

6.1.5 Stopping Criteria

ALNS searches through the solution space for many iterations. The stopping criterion decides the number of iterations performed in ALNS. Below we describe the different stopping criteria in this thesis.

Max Iterations

Max iterations is a straightforward stopping criterion, and the only thing needed is to keep track of how many iterations ALNS has performed. When the specified number of iterations is reached, the stopping criterion returns true, and the search is finished.

Not Accepted Count

A slightly more sophisticated stopping criterion is what we name not accepted count. The not accepted count stopping criterion terminates ALNS when a given number of iterations without having an accepted solution is reached. The counter resets to zero every time a solution is accepted. The rationale behind this stopping criterion is the belief that a local optimum is reached when all candidate solutions have been rejected for a given number of iterations, and there is no point in continuing the search.

6.1.6 Adaptive Weights and Operator Selection

ALNS selects an operator each iteration according to a probability distribution based on the weights. The probability distribution is given by

$$P(O = o) = \frac{w_o}{\sum_{i=1}^{N} w_i}, \tag{6.3}$$

where O is a random variable, w_o is the weight for operator o and N is the number of operators. The weight for the selected operator is updated at the end of each iteration based on how well the operator performed. The weight update is given by

$$w_{o,t+1} = (1 - \alpha) \cdot w_{ot} + \alpha \cdot r_t, \tag{6.4}$$

where w_{ot} and $w_{o,t+1}$ are the weights for the selected operator o in iteration t before and after the update, respectively. The reaction factor, α, controls how we weigh the previous weight relative to the newly received reward, r_t. The reward is based on the performance of the operator, which Section 6.1.7 elaborates on. If α is set to zero, the weights will not be updated at all, and if α is one we only care about the most recently received reward. Section 2.5.4 briefly describes the learning rate used for training an ANN, and the reaction factor in ALNS can be easily compared.

The weight updates and stochastic selection of operators make it so that at any given point in time during ALNS, the probability of selecting well-performing operators is

higher than the probability of selecting operators that have performed worse. This is with the belief that operators that have previously performed well will continue to perform well. If not, their weights will decrease, and so will their probability of being selected.

6.1.7 Reward Function

After an operator is selected and applied to a solution in ALNS, the new candidate solution can either be rejected or accepted. Based on whether the solution was accepted or not and how good the new solution is if accepted, we use the following reward function to reward the performance of the selected operator:

$$r(x_{current}, a, x'_{current}, x_{best}) = \begin{cases} 5 & \text{if } f(x'_{current}) < f(x_{best}) \\ 3 & \text{if } f(x'_{current}) < f(x_{current}) \\ 1 & \text{if } a = 1 \wedge x'_{current} \neq x_{current} \\ 0 & \text{otherwise} \end{cases}. \tag{6.5}$$

$x_{current}$ is the current solution before the selected operator is applied, $f(x_{current})$ is the corresponding objective value, $x'_{current}$ is the current solution after the selected operator is applied, x_{best} is the best solution encountered thus far, and $a = 1$ and $a = 0$ indicate that the candidate solution is accepted and rejected, respectively. The reward is used to update the weight for the selected operator, as Section 6.1.6 describes.

The highest reward of five is rewarded if the new solution has a better objective value than the best solution known at that iteration, which makes sense as the ultimate goal for ALNS is to terminate with as good a solution as possible. The second highest reward of three is rewarded if the new current solution has a better objective value than the previous. Lastly, the reward function distinguishes between cases where the candidate solution is accepted and rejected. If a solution x is accepted, but the objective is worse than the previous solution, we still want to reward the operator that resulted in x. Even though it is not a better solution, the fact that it is accepted means that the search is progressing. The reward function will not give a reward other than zero if the accepted solution is the same as the previous solution.

6.1.8 Feasibility

Since the optimal solution might not be reachable from the initial solution without traversing infeasible neighbors, we allow ALNS to accept infeasible solutions when searching. Restricting the traversal of the neighborhood to only the feasible region could remove good potential paths toward better solutions. Therefore, we relax some hard constraints, heavily penalize infeasible solutions and let ALNS traverse the neighborhood as usual. This means that the acceptance criterion can accept infeasible solutions. However, as they are heavily penalized, they are less likely to be improving solutions and are thus less likely to be accepted, dependent on the

acceptance criterion. When a new best solution is found, it is ensured that it is feasible.

Only some hard constraints are relaxed. This is due to implementation limitations and what we consider appropriate. We do not, for example, allow an employee to cover more than one unit of demand simultaneously, as this could prove challenging to repair. We also keep constraints that enforce the correct value on auxiliary variables. The relaxed hard constraints are daily- and weekly rest rules, blocked hours, and demand coverage; in other words, the rest- and demand constraints we outline in Section 5.6. Violations of these constraints are counted and weighted by individual *violation weights*. We set these weights an order of magnitude above other weights to incentivize finding feasible solutions. Setting this weight is a tradeoff between incentivizing exploration of the infeasible region and finding feasible solutions.

The new objective is given as

$$\min z = \sum_{e \in \mathcal{E}} f_e \tag{6.6a}$$

$$+ W^F A^F g \tag{6.6b}$$

$$+ \sum_{t \in \mathcal{T}} V_t^T \sum_{c \in \mathcal{C}} (W^{D+} \delta_{tc}^+ + W^{D-} \delta_{tc}^-) \tag{6.6c}$$

$$+ W^{DV} \nu_d + W^{RV} \nu_r + W^{OV} \nu_o + W^{BV} \nu_b, \tag{6.6d}$$

where ν_d, ν_r, ν_o and ν_b denote the number of demand, daily- and weekly rest and blocked hours violations with corresponding weights, respectively. The violation variables are non-negative.

6.2 Neural Network Assisted Large Neighborhood Search

This section describes a modification to ALNS where the adaptive weights are removed. The operator selection is performed using an ANN which is pre-trained using a version of the RL algorithm PPO. The network receives a feature vector representing the state and outputs a probability distribution over the operators. We call this algorithm NNALNS. The reasoning is that the ANN can parameterize a better selection strategy than the adaptive weights used in ALNS. In fact, a meta-study by Turkeš et al. (2021) found that the adaptiveness of ALNS only gives a 0.14% improvement over the non-adaptive version with a constant uniform distribution.

Firstly, Section 6.2.1 outlines the algorithmic framework for NNALNS. Following, Section 6.2.2 describes the different features that constitute the feature vector used as input for the ANN. Lastly, Section 6.2.3 describes how the ANN is trained and provides details for the specific PPO algorithm used during training. See Section 2.5 for a brief walkthrough of RL.

6.2.1 Algorithmic Framework

Figure 6.6 illustrates how NNALNS works, and by comparing the figure to Figure 6.1 it should be clear the the differences between NNALNS and ALNS are minimal. NNALNS is an extension of the LNS framework, where several sub-heuristics, or operators, compete during the neighborhood search; that is also true for ALNS, and the main difference lies in the adaptive weights of ALNS. ALNS updates weights for the operators during a run of the algorithm and selects an operator in each iteration according to the probability distribution given by the weights. On the other hand, NNALNS uses a pre-trained ANN to output the probability distribution in each iteration given a feature vector representing the state. That is, we can somewhat loosely say that ALNS learns on the go, whereas NNALNS learns prior to a run.

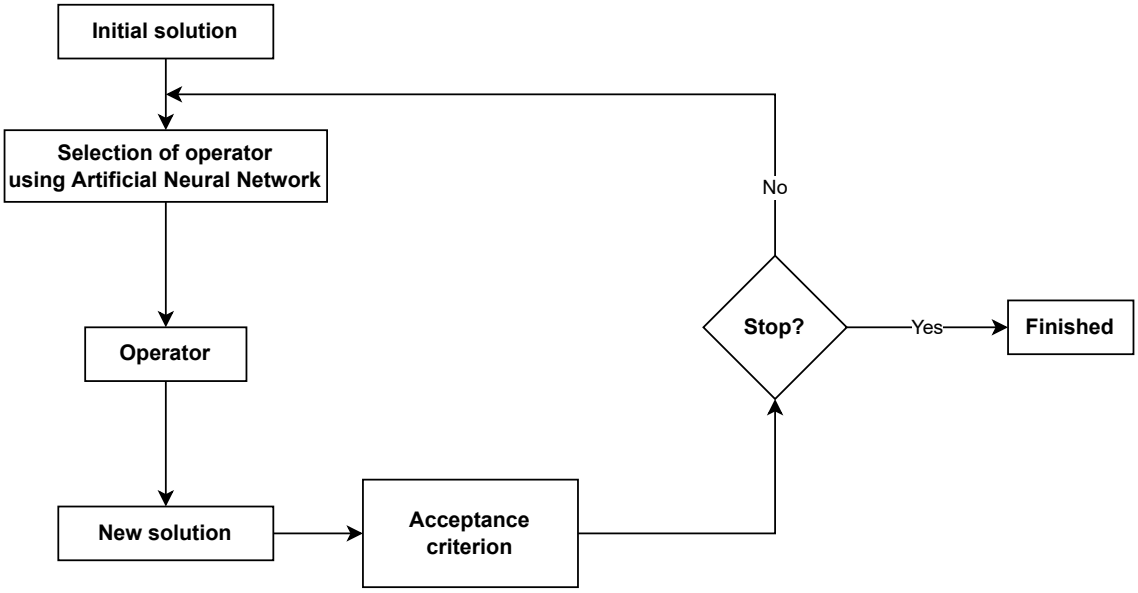


Figure 6.6: Algorithmic sketch of NNALNS.

Algorithm 9 provides the pseudocode for NNALNS. The input is an initial solution, x_0, which is found solving the MIP model described in Section 6.1.2, and a policy, $\pi_{\boldsymbol{\theta}}$, parameterized by the weights of an ANN, $\boldsymbol{\theta}$. The policy describes for any given state during the search, s, the probability of selecting operator o as $\pi(o|s, \boldsymbol{\theta})$. In Line 4 a feature vector representing the state is extracted, and in the following line an operator is selected according to the probability distribution

$$P(O = o) = \pi(O = o|s, \boldsymbol{\theta}), \tag{6.7}$$

where O is a random variable, and o is a particular operator. Following the operator selection, the selected operator is applied to produce a candidate solution that is accepted or rejected according to the acceptance criterion. This is no different from ALNS, but there is no need to update the weights for the selected operator based on the performance; the policy already represents a mapping from any given state to the corresponding probability distribution over the operators. Selecting an operator, finding a new candidate solution, and using the acceptance criterion to accept or reject the candidate solution is repeated until the stopping criterion is met. Section 6.1.5 describes different stopping criteria for ALNS and NNALNS.

Algorithm 9 NNALNS

Input: x_0: initial solution, $\pi_{\boldsymbol{\theta}}$: policy
Output: x_{best}: best solution found

$x_{current} := x_0$
$x_{best} := x_0$
while stopping criterion not met **do**
    $s :=$ GetFeatureVector($x_{current}$)
    $o :=$ SelectOperator($\pi_{\boldsymbol{\theta}}, s$)
    $x_{candidate} := o(x_{current})$
    if Accept($x_{candidate}, x_{current}$) **then**
        $x_{current} := x_{candidate}$
    end if
    if $f(x_{current}) < f(x_{best})$ **then**
        $x_{best} := x_{current}$
    end if
end while
return x_{best}

6.2.2 Feature Vector

We encode the state of the RRP as matrices of shift assignments and demand coverage. Introducing such a state encoding would result in a substantial input layer for an ANN, which could increase the training required. Most importantly, features like the raw schedule encoding would result in feature vectors of varying lengths between instances. As this would exclude learning across instances, such a feature vector is unsuitable. Therefore, we propose a feature vector consisting of problem-specific and search-based attributes of constant size, intended to describe both the relevant qualities of a current solution and the state of the search.

Search-Based Features

The first part of the feature vector consists of search-based features, as we classify in Section 3.4.5. Such features describe the state of the search algorithm without introducing problem-specific concepts. These are appropriate for several reasons. First of all, they are analogous to the information available to an ALNS. As NNALNS can be considered a modification of ALNS, and these features have proven effective for ALNS, they should be reasonable for NNALNS as well. Secondly, by definition, such features are more generalizable than problem-specific ones. While feature ranges and distributions may vary considerably between problems or even instances, they do not need to be tailored according to the problem definition and are thus more general. Search-based features are far from the key to generalizability but nevertheless good attributes for a feature vector.

The goal of search-based features is to facilitate high-level strategies regarding the search procedure. Such strategies can be insights like detecting and handling local optima, automatically evaluating the exploration/exploitation tradeoff, or assessing

Table 6.1: Search-based features.

Feature	Description
reduced_cost	The difference in objective value compared to the previous solution
cost_from_min	Difference in objective value compared to the currently best solution
min_cost	The currently best objective value
cost	The current objective value
no_improvement	The number of iterations since last improvement of objective value
is_legal	Whether the current solution is feasible
index_step	The iteration number
was_changed	Whether the last operator changed the solution. 1 if true, 0 otherwise
unseen	Whether the current solution has been encountered before, 1 if true, 0 otherwise
last_action_sign	Whether the previous operator improved the solution, 1 if true, 0 otherwise
last_action	The previous operator applied, 1-hot encoded
ac_status	The status of the acceptance criterion. Threshold value for threshold acceptance, temperature and cooling schedule for simulated annealing

the appropriate degree of change for a particular search state. Our selection of search-based features is inspired by Kallestad (2021) and can be found in Table 6.1.

Many diverse selections of features could be justified, as their impact on the ANN's ability to learn search strategies is complex and composite. Therefore we suggest some general qualities that one should be able to determine from the state intuitively and choose features to cover these qualities:

- Total search state - how well the entire search process has fared thus far.
- The local search state - the current trajectory of the search and local optima.
- Minimize repeated or unnecessary operator choices.

The intent of the ANN is to learn such structures automatically, but considering such areas could facilitate a suitable state representation. The majority of the attributes are dedicated to the local search state, as *reduced_cost, cost_from_min, cost, no_improvement, is_legal* and *last_action_sign* all describe current changes and attributes related to local optima. The total search state is described by *min_cost, index_step* and *ac_status*, as they convey how far along the search is and the current best solution. Finally, *was_changed, unseen* and *last_action* are intended to stop the ANN from repeatedly choosing operators without positive impact. Features where maximum and minimum values are known are normalized. Where this is impossible, we scale them according to problem size to achieve values between zero and one.

Problem-Specific Features

Table 6.2: Problem-specific features.

Feature	Description
assigned_ratio (4)	Percentage of days each employee is assigned to a shift; mean, std, max, min
uncovered_demand_ratio (1)	Percentage of days that have uncovered demand
individual_cost_ratio (4)	Individual employees' fairness costs, compared to total fairness cost; mean, std, max, min
weekly_cost_ratio (4)	Individual weeks' fairness costs, compared to total fairness cost; mean, std, max, min
fairness_aspect_ratios (8)	Ratio of different fairness aspects' contribution to total fairness cost
fairness_ratio (1)	Ratio of the total fairness cost compared to total objective value

The other part of the feature vector consists of problem-specific features, which are meant to provide the ANN with insight to develop lower-level strategies. While the search-based features are intended to decide how the search should be conducted, the problem-specific features are instead intended to determine how to best improve a current solution. To achieve this, we present some requirements that we believe a feature vector should describe:

- Solution structure
- Complexity - how difficult it is for the operators to satisfy different constraints in the current solution
- Objective function relatability - features describing the current cost of the different parts of the objective function.

Aside from these requirements, we introduce two restrictions on the problem-specific features. The first is that we must ensure that the feature vector is of constant length, and the second is that individual features are of the same scale regardless of instance size. The first restriction disallows features akin to "fairness cost per employee" as the number of employees varies between instances. Instead, we use the mean, maximum, minimum, and standard deviation of aspects like individual fairness and weekly costs as features. The second restriction requires the features to be ratios instead of accumulated values. Normalization is not possible for the problem-specific features as the maximum and minimum values are not known. We instead use ratios, as described in Table 6.2.

Table 6.2 describes the problem-specific features. As there is no way to represent individual shift assignments without breaking the first restriction mentioned

above, we use the *assignment_ratio* to describe the schedule structure. *uncovered_demand_ratio*, *individual_cost_ratio* and *weekly_cost_ratio* handle complexity, as more of these costs indicate a difficulty in scheduling different parts of the problem. The same features, together with the *fairness_aspect_ratios* and *fairness_ratio*, also cover the objective since they make up the different parts of the objective function.

6.2.3 Training

The premise for NNALNS to perform well is that the parametrization of the policy, $\pi_{\boldsymbol{\theta}}$, represents a good mapping from any state, s, to a probability distribution over the operators. To accomplish this, the ANN that parameterizes the policy needs to be trained. In this thesis, we use a version of PPO that uses the actor-critic architecture to train the ANN. Section 2.5 provides the necessary foundation for RL to understand this section.

Algorithm 10 describes how the ANN that parameterizes the policy is trained. The input for the training algorithm is an environment, e, the number of weight updates for the ANN, U, the number of epochs per weight update, N, and the number of episodes between each update, E. The environment, in this case, is an abstract object that keeps track of the state of the search. It keeps track of things such as the current solution, the best solution, and the initial solution and can therefore be viewed as ALNS without the weights. The RL agent learns by repeating the process of selecting an operator, applying it to the environment, and perceiving the reward and new state.

Line 1-4 initialize the weights for the actor-ANN and the critic-ANN. Following, the training algorithm will perform E episodes before each update. An episode is defined as a complete run of NNALNS until the stopping criterion is met. The agent (i.e., actor and critic) interacts with the environment by applying an operator. Algorithm 11 describes how the environment changes as a consequence of the agent's interaction. After E episodes, advantages are calculated as

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{6.8}$$

where $\delta_t = r_t + \gamma V_{\boldsymbol{\phi}_{old}}(s_{t+1}) - V_{\boldsymbol{\phi}_{old}}(s_t)$, t is the iteration number within a single episode, and r_t is the perceived reward from choosing and applying an operator in iteration t. Section 6.1.7 describes the reward function for ALNS, and the same is used for training the policy for NNALNS to get a fair comparison. The discount factor, γ, and exponential weight discount, λ, are numbers in the interval $[0, 1]$. δ_t is assumed to be zero when t is greater than the length of the episode, and $V_{\boldsymbol{\phi}_{old}}(s_t)$ is the value estimate for state s_t from the value function parameterized by the critic-ANN. Equation (6.8) corresponds to the generalized advantage estimate, which was proposed by Schulman et al. (2015). Before updating the weights of the actor and critic, the discounted cumulative rewards are also calculated according to

$$R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}, \tag{6.9}$$

where r_t is assumed to be zero if t is greater than the length of the episode.

The advantages and discounted cumulative rewards are a part of the loss function used to calculate gradients to update the weights in the actor-ANN and critic-ANN. The loss function we use in this thesis is given by

$$L(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\min(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t) \tag{6.10a}$$
$$+ \beta(V_{\boldsymbol{\phi}}(s_t) - R_t)^2 \tag{6.10b}$$
$$- \mu H(\pi_{\boldsymbol{\theta}}(s_t)), \tag{6.10c}$$

where (6.10a) is the L^{CLIP} loss for the actor which Section 2.5.5 describes, (6.10b) is the mean squared error loss for the critic, and (6.10c) is an entropy bonus for the actor. β and μ are used to scale the size of the critic's loss and entropy bonus relative to the L^{CLIP} loss. The entropy bonus is a part of the loss function because we want to incentivize the actor-ANN to output a more unpredictable probability distribution; we want the agent to explore the consequences of choosing different operators instead of converging too quickly to a local optimum.

After the agent has repeated the process of interacting with the environment for E episodes and then updating the weights for N epochs U times, the training algorithm terminates and returns the parameters of the actor-network and critic-network. The parameters of the actor-network parameterize a policy, which can be used for the NNALNS algorithm. Although we use the same reward function in Algorithm 10 and ALNS, it is critical to note that the ultimate goal in RL is to maximize the cumulative reward. The consequence of this is that though the agent can learn how to receive a higher cumulative reward than ALNS, this does not necessarily mean that NNALNS performs better than ALNS in terms of objective value. When an RL agent learns to achieve high returns in an unexpected way without achieving the goal intended by the designer of the reward function, it is referred to as *reward hacking* (Yuan et al., 2019).

Algorithm 10 Training

Input: e: environment, U: number of updates, E: episodes per update, N: number of epochs

Output: $\boldsymbol{\theta}$: parameters of learned policy (actor), $\boldsymbol{\phi}$: parameters of learned value function (critic)

Initialize the parameters for the actor ANN, $\boldsymbol{\theta}$
$\boldsymbol{\theta}_{old} := \boldsymbol{\theta}$
Initialize the parameters for the critic ANN, $\boldsymbol{\phi}$
$\boldsymbol{\phi}_{old} := \boldsymbol{\phi}$
for update $:= 1, 2, \ldots, U$ **do**
    $T := 0$
    for episode $:= 1, 2, \ldots, E$ **do**
        $s :=$ RESET(e)
        while episode is not finished **do**
            Select operator o according to policy $\pi_{\boldsymbol{\theta}_{old}}$
            $s_{new}, r :=$ STEP(e, o)
            BUFFER(s, r, o)
            $s = s_{new}$
            $T := T + 1$
        end while
    end for
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$ and discounted rewards $R_1, \ldots, R_T$
    Train actor-ANN ($\boldsymbol{\theta}$) and critic-ANN ($\boldsymbol{\phi}$) for N epochs
    $\boldsymbol{\theta}_{old} := \boldsymbol{\theta}$
    $\boldsymbol{\phi}_{old} := \boldsymbol{\phi}$
end for
return $\boldsymbol{\theta}$, $\boldsymbol{\phi}$

Algorithm 11 Step

Input: e: environment, o: operator

Output: s_{new}: new state

$x_{current} =$ GETCURRENTSOLUTION(e)
$x_{candidate} := o(x_{current})$
acc $:=$ ACCEPT$(x_{candidate}, x_{current})$
if acc **then**
    $x_{current} := x_{candidate}$
end if
if $f(x_{current}) < f(x_{best})$ **then**
    $x_{best} := x_{current}$
end if
$s_{new}, r :=$ UPDATESTATE$(e, x_{current}, x_{best}, \text{acc})$
return s_{new}, r

Chapter 7

Test Instances and Parameters

In this chapter, we present the test instances and parameter decisions used in Chapter 8. Firstly, Section 7.1 provides a discussion of the characteristics of the test instances and their implications for NNALNS and ALNS. Following, Section 7.2 presents our choice of important parameter values.

7.1 Test Instances

We conduct the tests of NNALNS and ALNS on 13 problem instances. Approximately half of them are real-life problem instances provided by Visma Resolve, referred to as Visma instances. Parts of the problem definition of the RRP, for example, competence requirements, are derived directly from the structure of the instances in this dataset.

The other half of the test instances are derived from the publicly available NRP benchmark dataset (*Nurse Rostering Benchmark Instances*, 2014). We refer to them as the Benchmark instances. Many works within the NRP community use this dataset to benchmark the performance of solution algorithms on NRPs (Brucker et al., 2010; Asta et al., 2016; Smet, 2018). Since the instances are designed for a particular NRP (Burke et al., 2008), we modify them to fit our problem definition while maintaining the characteristics of the different instances. We include the benchmark data to test ALNS and NNALNS on a broader range of instances to conduct a sufficient number of tests. In addition, the benchmark instances are sorted somewhat systematically regarding problem size, allowing us to assess the algorithms' performance on an increasing instance size.

Table 7.1 lists all the instances included in the computational study, along with their key characteristics. We name instances after the dataset they stem from: 'B' for Benchmark and 'V' for Visma. The asterisk symbol (*) denotes instances derived from altering other test instances, such as adding extra employees or days. Furthermore, the naming convention includes the number of days and employees in the instance, resulting in the format 'V/BX-dXX-eXX'.

Table 7.1: Characteristics of test instances.

Instance	Days	Employees	Shift-types	Demand intervals/day
B1-d14-e14	14	14	2	2
B2-d28-e16	28	16	2	2
B3-d28-e50	28	50	6	4
B4-d42-e45	42	45	6	10
B5-d56-e20	56	20	3	4
B6-d84-e22	84	22	3	4
B7-d182-e50	182	50	6	4
V1-d56-e9	56	9	31	24
V2-d70-e6	70	6	13	19
V3-d46-e28	46	28	30	17
V4-d84-e8	84	8	4	5
V5-d42-e15	42	15	15	20
V6-d98-e15* (V5)	98	15	15	20

Instance Size

Problem size can, to some degree, be assessed from the descriptive data. Here, size refers to an approximate assessment of the number of variables and constraints in a MIP model. The number of days and employees describes the length of the planning horizon and the number of employees that can be assigned to a shift each day of this period. Therefore, this is a reasonable estimation of the problem size. Another size measure is the number of shift types, denoting the number of possible assignments each day for each employee.

The problem instances provide a wide range of sizes, from two weeks to six months, from eight to 100 employees, and from one to 30 shifts each day. The benchmark data is, in these terms, more varied. Both datasets contain about the same number of days on average, but the benchmark data contains, on average, more than double the number of employees, although some outliers exacerbate this. However, the Visma data contains considerably more shift-types.

Instance Complexity

On the other hand, complexity can be hard to assess from the descriptive data alone. Several factors come into play when deciding how hard a problem instance is to solve and are often more difficult to determine from statistics. An example of such a factor is demand structure, where many aspects can increase complexity. The legal deviation from ideal demand is an essential factor, and in most real-life instances, the minimum and ideal coverage are identical. The magnitude of demand requirements is also important for complexity, especially compared to the number of employees. While not a perfect measure, the number of demand intervals can indicate how difficult it is to cover demand. Note that this is not a description of how much demand there is, but rather how demand varies throughout the day. In this regard, the Visma data seems complex. Combining this with the fact that the

Visma data also contains more shift-types, the Visma instances appear much more difficult to solve. Preliminary testing supports this hypothesis.

7.2 Parameters

This section outlines the most important parameters used during the computational study and reasoning behind our parameter choices. Parameter configuration highly affects the behavior of the system. However, the number of parameters concerning both ALNS and NNALNS is large. Thus, the number of possible parameter combinations quickly explodes in size. As our thesis is for the most part concerned with the operator selection in the ALNS framework, we use the same parameters for NNALNS and ALNS. Many of the parameters are decided based on preliminary testing.

7.2.1 Problem Parameters

The problem parameters include the weighting of the various costs included in the objective function of the problem, described in Section 5.3. These weights highly affect the structure of the solutions produced by the algorithm. As we are comparing operator selection strategies and not whether the solutions produced are satisfactory in a real-world setting or not, we do not investigate the effect of applying different cost weight configurations. Instead, the cost weights are set according to preliminary studies of the RRP (Langfeldt et al., 2021). Simple preliminary tests show that these are reasonable weights as they give a balanced cost distribution between the different types of costs. A table of the cost weights is included in Table B.5 in Appendix B.

The violation weights described in Section 6.1.8, ensures traversal in the infeasible region for ALNS and NNALNS. The weights need to be set to values that balance the allowance of infeasible solutions but favor the feasible ones. We set these weights to 100. Observations made during preliminary testing showed that with these values, both infeasible and feasible solutions were included in ALNS and NNALNS, with the majority being feasible.

7.2.2 NNALNS and ALNS Parameters

NNALNS and ALNS can be configured in various ways as shown in Chapter 6. This section provides the configurations of NNALNS and ALNS held constant during the computational study.

Reaction Factor

We set a value of 0.3 for the reaction factor, α, used to update the weights of the operators in ALNS. The rationale is that we believe this to be a reasonable weighting

between previous rewards and the immediate reward.

Operators

Section 6.1.3 outlines the different types of operators we develop for the RRP. Each destroy operator can be paired with each repair operator and vary in size. We employ destroy-repair pairs with fixed destruction sizes. In addition, we include all of the hybrid operators presented in Section 6.1.3, with a pre-set size. In total, this yields 27 operators. Tables B.3 and B.4 list the operators we use during the computational study.

Kallestad (2021) conclude that heuristic selection using ANNs perform better with an increased number of heuristics/operators compared to ALNS. However, preliminary testing indicates that this might not be the case for NNALNS when applied to the RRP, using complex destroy-repair operators and hybrid operators. Thus, we limit the number of operators to the ones presented in Appendix B.1.

Stopping Criterion

To be able to compare NNALNS with other operator selection strategies at the same iteration stage, we use *max iterations* as the stopping criterion in our computational study. We set the maximum number of iterations to 1000. Preliminary tests indicate that the best objective value converges before 1000 iterations for many problem instances. Therefore we limit ourselves to measuring performance after 1000 iterations during the computational study.

Acceptance Criterion

We have explored different acceptance criteria during preliminary testing. Among *hill climbing*, *simulated annealing* and *threshold acceptence*, threshold acceptance showed the most promising results. Based on these observations, we use threshold acceptance in the computational study. The linear decay of the threshold is set to 0.0025 so that only improving solutions are accepted after 400 iterations. As we set the stopping criterion to 1000 iterations, the exploration phase of the search is limited to 40% of the total number of iterations. By exploration phase, we mean the iterations in which the acceptance criterion can accept non-improving solutions. Table 7.2 summarizes the acceptance criterion parameters.

Table 7.2: Acceptance criterion parameters.

Acceptence Criterion	
Type	Threshold acceptance
Initial threshold (T_0)	1
Linear decay (k)	0.0025

Reward Function

We use the reward function described in Section 6.1.7. Our rationale is that by using the same reward scheme for NNALNS and ALNS, it is possible to compare cumulative rewards obtained by them. This allows us to evaluate the policy learning ability for NNALNS, as the objective of RL is to maximize accumulated rewards without directly considering our ultimate goal of finding the best possible solutions in terms of objective value from an OR point-of-view.

Features

Table 6.1 and Table 6.2 show all the features available for constructing a feature vector representing the state in NNALNS. In the computational study, we use either only the search-based features in Table 6.1 or both the search-based features and problem-specific features in Table 6.2. By doing this, we can compare the performance of NNALNS with and without the problem-specific features.

Network Architectures

We use two ANNs during training of the policy representing the operator selection strategy in NNALNS: one for the actor and one for the critic. Both networks are standard feedforward ANNs with three fully connected hidden layers. The actor-network includes an additional softmax layer on the output, to ensure that the output values of the network sums to one to obtain a valid probability distribution. Table 7.3 summarizes the network architectures.

Table 7.3: Parameters for the neural networks in NNALNS.

	# Hidden Layers	**Activation Function**
Actor	3	Tanh
Critic	3	Tanh

Training Parameters

The training parameters affect the learning abilities of the actor-network and critic-network. Table 7.4 shows the most important parameters used to train the policy for NNALNS. Preliminary testing shows that the average cumulative reward per episode between each update converges after around 50 updates for several instances. Accordingly, we limit the number of updates to 50. Furthermore, we run 20 episodes between each update to ensure that enough data is captured. This results in a total of 1000 episodes during training. The learning rates for the actor-network and critic-network are challenging parameters to set, as the optimal learning rate is situation-dependant. We decide the learning rates based on empirical knowledge. As we observed that the average cumulative reward increases after each update on all instances during preliminary testing, we regard the learning rates as satisfactory

for this thesis. Furthermore, we use the Adam optimizer for training the ANNs, which has shown to be a high-performance optimizer for PPO (Schulman et al., 2017). More technical parameters related to PPO can be found in Table B.6 of Appendix B.

Table 7.4: Training parameters used during training of policies for NNALNS.

Training parameters	
Total # of episodes	1000
Episodes before update	20
Number of updates	50
Learning rate actor	0.0003
Learning rate critic	0.001
Optimizer	Adam
Epochs	50

Generalized Training Parameters

For generalized training on instance permutations, we necessarily train the policies on more than one permutation. Unrelated parameters remain unchanged, but the number of episodes is adjusted. In order to run a sufficient number of episodes per instance, we increase the number of episodes before each update to 32, resulting in eight episodes per permutation per update, or 1600 in total. Preliminary testing indicates that eight episodes per instance is a reasonable trade-off between computational time and sample size.

For generalized training across problem instances, we select three instances per dataset. This allows us to choose varied training instances while leaving multiple test instances for evaluation. We run ten episodes per instance to maintain a similar computational time.

Table 7.5 presents the changed parameters for generalized learning. Parentheses indicate the parameters for training across problem instances.

Table 7.5: Training parameters used during generalized training of policies for NNALNS.

Training parameters	
Total # of episodes	1600 (1500)
Episodes before update	32 (30)
Number of updates	50
Number of instances	4 (3)
Episodes per instance	8 (10)
Learning rate actor	0.0003
Learning rate critic	0.001
Optimizer	Adam
Epochs	50

Chapter 8

Computational Study

This chapter presents and discusses computational results from using ALNS and NNALNS on the instances of the RRP which we describe in Section 7.1. Firstly, Section 8.1 provides a overview of the hardware and software specifications for the test environment, before Section 8.2 describes how the tests are performed and motivates them based on the goals we set for this thesis in Chapter 1. Following, Section 8.3 discusses the value of adding problem-specific features to the feature vector used in NNALNS. Section 8.4 provides a comparison of ALNS and NNALNS, before Section 8.5 discusses selection strategies learned for NNALNS and compares them to the selection strategies resulting from ALNS. Finally, Section 8.6 evaluates NNALNS on its ability to generalize across problem instances.

8.1 Test Environment

We perform all tests on the Department of Industrial Economics and Technology Management's high-performance cluster, Solstorm, with the software and hardware specifications in Table 8.1 to ensure fair comparisons.

Table 8.1: Hardware and software specifications for the test environment.

CPU	4 x AMD Opteron 6274
Cores / Frequency	64 / 2.2GHz
RAM	128 GB
Operating System	CentOS Linux 7
Python version	3.9.5
Gurobipy version	9.1.2
PyTorch version	1.10.2
MPIRE version	2.3.0

We have written all implementations in this thesis in the programming language Python. Gurobi is a commercial optimization solver used to solve the MIP model we use for finding initial solutions, as described in Section 6.1.2, and Gurobipy is

the Python interface for this. During the policy training for NNALNS multiple episodes are run in parallel before each update, utilizing the available CPU cores. The parallelization is implemented using the fast and user-friendly multiprocessing library MPIRE. We have implemented and trained all of the ANNs using the Python package PyTorch. Lastly, the problem instances are provided in Extensible Markup Language (XML) before being parsed into Python objects for further use.

8.2 Experimental Setup

The four overarching goals for this thesis are the motivation behind all of the different experiments we perform in the computational study. We want to investigate whether NNALNS can learn more intelligent operator selection strategies for the RRP compared to ALNS, include an enriched state representation of an RRP solution for NNALNS, examine the generalizability of learned selection strategies across different RRP instances, and develop operators tailored to the RRP targeting different properties of a solution.

For the results discussed in the computational study we train two policies for each problem instance according to the training algorithm (Algorithm 10). This is because we want to have one policy trained with both search-based features and problem-specific features and one policy trained only with the search-based features for comparison.

After each update in Line 18 of Algorithm 10 we obtain a version of the previous policy where the parameters have been adjusted according to the data from the intermediate episodes. Although the hope is that the policy improves after each update, this is not guaranteed. Therefore, each result presented from applying NNALNS to an instance is affected by the policy that is chosen as input. When we want to discuss the performance of NNALNS concerning the objective value, we select the policy that, on average, gave the best objective value during training. On the other hand, we select the last policy obtained from training when we want to discuss performance in terms of cumulative reward. This is because preliminary testing shows that the last policy, on average, accumulates the most rewards. In contrast, the policy that performs best in terms of objective value needs to be assessed individually for each instance and training configuration (i.e., using all features or only the search-based features). Table 8.2 summarizes how we select policies according to the subject of interest for each instance. For further detail on policy selection, see Appendix C.5.

Table 8.2: How we select policies for NNALNS.

Subject	Strategy
Objective value	Choose the policy that, on average, gave the best objective value
Reward	Choose the last policy

When we compare the performance of NNALNS and ALNS, we also test a version of ALNS that does not adaptively update the weights after each iteration as a baseline.

This version of ALNS has a constant, uniform probability distribution over the operators, and we call it Uniform Large Neighborhood Search (ULNS). Section 8.3 discusses the differences between using only the search-based features and using all of the features to assess the value of including an enriched state representation of an RRP solution. In the comparative study in Section 8.4, we do not want to focus on the value of using the problem-specific features. Therefore, we perform training two times on each instance: one time using only search-based features and one including all features. We then, for each instance, select one policy for each of the two configurations according to Table 8.2, test both policies, and present the best result. The comparative study addresses whether NNALNS is able to learn more intelligent operator selection strategies for the RRP compared to ALNS or not. Furthermore, Section 8.5 takes a closer look at the differences between the operator selection strategies of NNALNS and ALNS.

Once the policies are selected, we test ULNS, ALNS and NNALNS with and without problem-specific features 20 times on each test instance to account for the stochasticity of the algorithms. The results we present in the computational study are the average of the 20 runs, and each run consists of 1000 iterations.

Generalized Learning

Section 8.6 evaluates NNALNS on its ability to generalize across instances. Section 8.6.1 discusses performance when NNALNS is trained on one set of instances, and then tested on another. This will be referred to as *cross-instance learning*. Section 8.6.2 evaluates the performance of NNALNS when trained on multiple modifications of one instance, and tested on the original instance. We refer to this as its *adaptability*. We test adaptability as it might be challenging to generalize strategies across significantly different instances, but more plausible to learn such strategies for a set of modified instances, which models realistic changes between planning periods. For these tests, we also run new tests of ULNS and ALNS as the initial solutions might be different compared to the previous tests. Generalizability is only tested within the datasets, as we expect a substantial difference between the two datasets. In other words, a policy trained on Visma instances will not be used to solve benchmark instances and vice versa.

For cross-instance testing, we divide the instances into training and test sets, see Table 8.3. This split is arbitrary but intends to represent a diverse selection of instances in each set. We run two sessions of training for each of the training sets, one with and one without problem-specific features, and select the best policies for the objective function, as in Table 8.2. We then test the policies on the test sets, and select the best policy for each dataset.

Table 8.3: Training and testing datasets.

Set	Visma	Benchmark
Train	V2, V4, V6	B1, B2, B3, B4
Test	V1, V3, V5	B5, B6, B7

To measure adaptability, we implement an algorithm to permutate instances. This algorithm can add employees and days to an instance or change the demand structure. The employees or days added are copies of random existing ones, while we change the demand structure by swapping demand between days. Table 8.4 shows the permutations used in our experiments. Each instance is altered to four different training instances, and the trained policies are tested on the original instance to run tests comparative with previous ones. Section 8.6.1 only considers the original Visma dataset. This is because we regard adaptability to be most interesting to examine in the case of real-world instances.

Table 8.4: Alterations of original instances.

Variation	Added employees	Added days	Changed days
1	4	7	0
2	0	0	7
3	2	0	7
4	2	7	7

8.3 Value of Problem-Specific Features

In this section, we present and discuss the differences between using NNALNS with policies trained with *only search-based features* and policies trained with *both search-based features and problem-specific features* on the test instances. Firstly, Section 8.3.1 looks at the differences in terms of objective value. Following, Section 8.3.2 discusses the differences in terms of cumulative reward. We refer to NNALNS without the problem-specific features as 'NNALNS w/o all features' and NNALNS with all features as 'NNALNS w/ all features' in the figures and tables.

8.3.1 Objective Value

Table 8.5: Summary of objective value improvement of NNALNS with and without problem-specific features over ULNS after 1000 iterations on all test instances, averaged over 20 runs. The best performing algorithm is emphasized in bold text.

	ULNS	**NNALNS w/o all feat.**		**NNALNS w/all feat.**	
Instance	Obj. val	Impr.(%)	Obj. val.	Impr.(%)	Obj. val.
B1-d14-e14	-85.948	**13.94**	-97.930	0.25	-86.164
B2-d28-e16	-72.714	**45.08**	-105.494	-35.86	-46.642
B3-d28-e50	11.947	**799.17**	-83.530	521.06	-50.304
B4-d42-e45	1351.447	**15.60**	1140.622	12.27	1185.646
B5-d56-e20	321.291	**57.600**	136.238	39.55	194.231
B6-d84-e22	423.651	**105.07**	-21.497	91.11	37.655
B7-d182-e50	2052.894	**62.64**	766.906	58.14	859.305
V1-d56-e9	996.674	**13.57**	861.453	12.26	874.508
V2-d70-e6	-244.357	**4.57**	-255.526	-0.93	-242.076
V3-d46-e28	-244.114	**86.52**	-455.322	83.57	-448.127
V4-d84-e8	787.358	**10.61**	703.825	9.83	709.932
V5-d42-e15	1015.204	**23.26**	779.038	18.88	823.520
V6-d98-e15	2619.869	17.14	2170.797	**28.53**	1872.520

Table 8.5 summarizes the results from using NNALNS with and without problem-specific features on all test instances, only considering the objective value improvement over ULNS. Figure 8.1 and Figure 8.2 show bar charts of objective value improvement over ULNS on benchmark instances and Visma instances, respectively. The results show that NNALNS performs better with only the search-based features on all test instances except V6-d98-e15. This might be surprising, as one would expect that a policy with problem-specific features has more information about the state. However, it is important to note that the policies are trained using RL; the ultimate goal in RL is to maximize the expected cumulative reward. Hence, a richer state representation may produce a better policy in terms of cumulative reward but without a gain in objective value. In fact, the enriched state representation may allow the RL agent to exploit structures of an instance to obtain more rewards at the expense of our ultimate goal to obtain the best possible objective value from an OR point-of-view. Another possible explanation is simply that more features require more time in training. This might be because the ANN needs more data to recognize the important features correctly, and an insufficient amount of data might make the ANN favor features that do not contribute much. Section 8.3.2 explores these possibilities further by looking at the cumulative rewards.

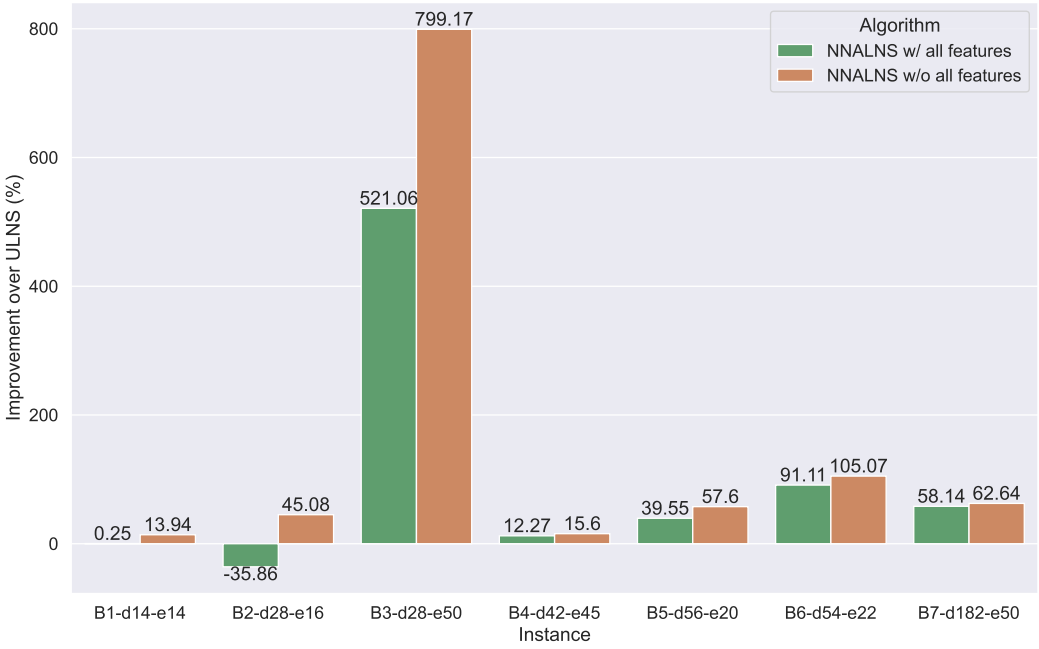


Figure 8.1: Bar chart of objective value improvement for NNALNS with and without problem-specific features over ULNS on benchmark instances after 1000 iterations, averaged over 20 runs.

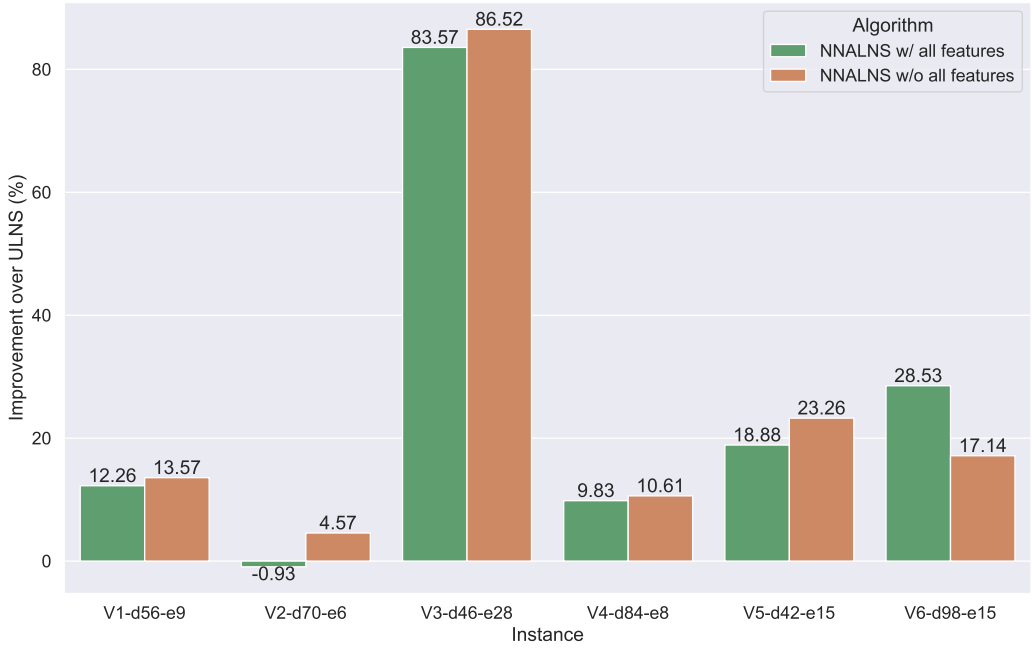


Figure 8.2: Bar chart of objective value improvement for NNALNS with and without problem-specific features over ULNS on Visma instances after 1000 iterations, averaged over 20 runs.

8.3.2 Reward

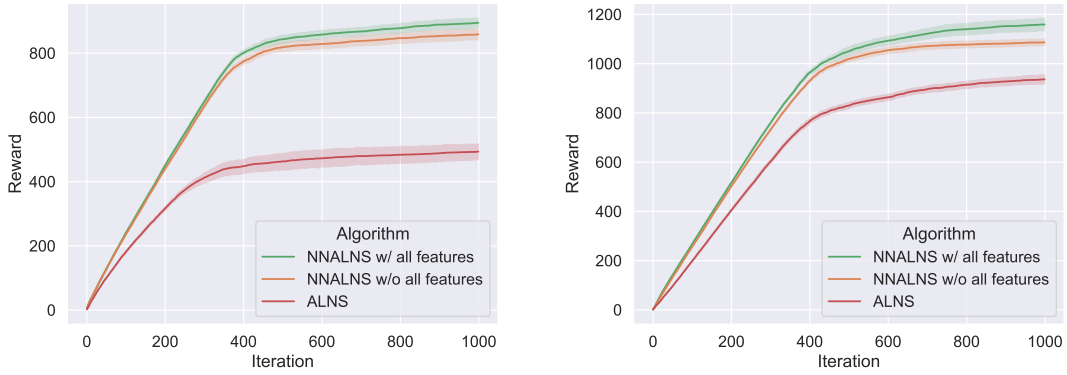
Table 8.6: Improvement of cumulative reward over ALNS for NNALNS with and without problem-specific features after 1000 iterations, averaged over 20 runs. The best performing algorithm is emphasized in bold text.

	Cum. reward	Improvement over ALNS (%)	
Instance	**ALNS**	**NNALNS w/o all feat.**	**NNALNS w/all feat.**
B1-d14-e14	238.85	**210.86**	207.14
B2-d28-e16	493.60	74.04	**81.22**
B3-d28-e50	454.15	103.77	**112.56**
B4-d42-e45	673.40	33.64	**41.17**
B5-d56-e20	622.60	41.33	**46.05**
B6-d84-e22	733.55	28.50	**28.57**
B7-d182-e50	869.60	**43.90**	42.98
V1-d56-e9	981.70	28.00	**28.08**
V2-d70-e6	585.50	**77.33**	66.40
V3-d46-e28	900.15	13.99	**36.32**
V4-d84-e8	788.95	22.82	**22.83**
V5-d42-e15	936.35	16.02	**23.80**
V6-d98-e15	1027.15	16.94	**22.64**

Table 8.6 shows the accumulated reward of ALNS and NNALNS with and without problem-specific features. Recall that rewards are given based on the same reward function per iteration for ALNS and NNALNS. The results show that for all instances except B1-d14-e14, B7-d182-e50 and V2-d70-e60, NNALNS with problem-specific features performs better in terms of cumulative reward. Furthermore, both NNALNS with and without problem-specific features obtain larger cumulative rewards on all instances. This implies that the training scheme in Algorithm 10 is successful from an RL point-of-view; by using RL, NNALNS is able to obtain more rewards than ALNS.

In Section 8.3.1, we observe that NNALNS without problem-specific features outperforms NNALNS with problem-specific features on almost all test instances in terms of objective value. However, the situation is quite different when looking at the cumulative reward. This observation points toward reward hacking; the addition of problem-specific features allows the policy to exploit structures of an instance to obtain more rewards at the expense of objective value. This explanation is more plausible than the explanation that more features require more time in training because we see that NNALNS with all features outperforms NNALNS with only search-based features regarding rewards. In light of this, the reward function itself is an interesting subject to explore for future research. A different reward function might obliviate the problem with reward hacking while using the enriched state representation as an advantage. Nevertheless, longer training times with all features is also an interesting direction to explore for future research.

Figure 8.3 shows plots of cumulative reward by using NNALNS with and without problem-specific features and ALNS on B2-d28-e16 and V5-d42-e15. An observation is that the cumulative reward seems to converge around 400 iterations. This makes sense, as the threshold in the acceptance criterion is zero at iteration 400. When the threshold is zero, the acceptance criterion accepts only improving solutions, which means that it is not possible to receive non-zero rewards unless the candidate solution is an improvement in objective value over the current solution. See Appendix C.2.2 for the cumulative reward plots for all test instances.



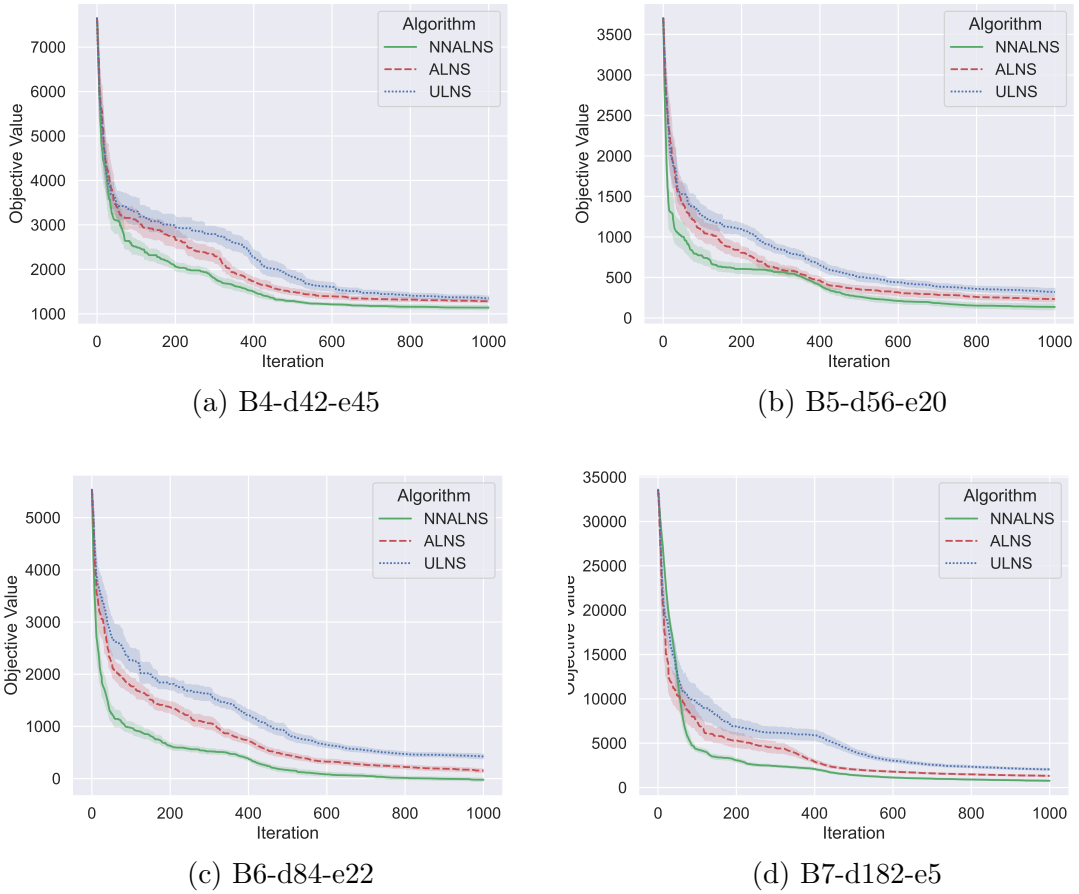
(a) B2-d28-e16

(b) V5-d42-e15

Figure 8.3: Average cumulative reward over 20 runs with a 95% confidence interval for two test instances using NNALNS with and without problem-specific features and ALNS.

8.4 Comparative Study

In the comparative study, we want to compare the performance of NNALNS and ALNS on different instances of the RRP. We use ULNS as a baseline and measure the relative improvement for ALNS and NNALNS after 1000 iterations. By doing this, we can gain insight into the value of having an adaptive strategy that changes throughout the search. Although NNALNS is not adaptive in the sense that the strategy, or policy, changes throughout the search, it is adaptive in the sense that the probability distribution over the operators is dependent on the state of the search and current solution.



(a) B4-d42-e45

(b) B5-d56-e20

(c) B6-d84-e22

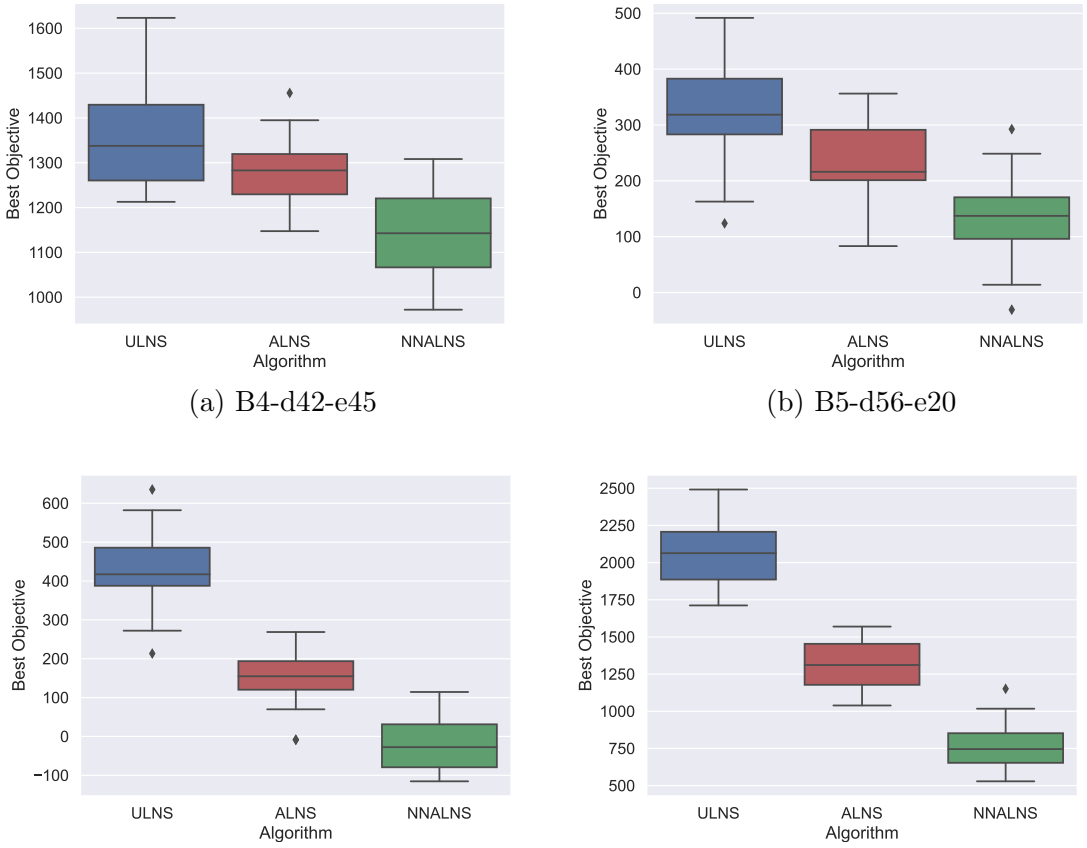
(d) B7-d182-e5

Figure 8.4: Average best objective value over 20 runs with a 95% confidence interval for four benchmark instances using NNALNS, ALNS and ULNS. The problem size increases from a) to d).

Figure 8.4 shows the evolution of the best objective value for NNALNS, ALNS and ULNS for 1000 iterations on a subset of the benchmark instances. We observe that the curves representing the best objective values during the search are downwards sloping for all algorithms. The fact that even ULNS, with a constant, uniform probability distribution over the operators, can improve the objective value in this way indicates that the operators we have developed can successfully improve the objective value by targeting different properties of an RRP solution.

The benchmark instances are designed to have increased complexity and an increased problem size with growing instance numbers. Thus, Figure 8.4 indicates that the relative performance of NNALNS over ALNS and ULNS tends to increases as the complexity and problem size increases, although the improvement of B7-d182-e5 is less than B6-d84-e22. It is hard to point out any similar pattern for the Visma instances, as the problem size and complexity are hard to assess for a Visma instance. To capture the spread, skewness, and variation of the distributions from the results based on 20 runs, Figure 8.5 shows box plots of the best objective value after 1000 iterations. We see that the boxes representing the interquartile ranges for NNALNS are below the boxes for ULNS and ALNS, which means that there is a significant

difference. All of the plots from the comparative study are present in Appendix C.1.



(a) B4-d42-e45

(b) B5-d56-e20

(c) B6-d84-e22

(d) B7-d182-e50

Figure 8.5: Box plot of best objective value for four benchmark instances after 1000 iterations of NNALNS, ALNS and ULNS over 20 runs. The problem size increases from a) to d).

Table 8.7 summarizes the results from the comparative study. On all instances, both NNALNS and ALNS perform better in terms of objective value. This indicates that it is, in fact, beneficial to have a probability distribution over the operators that varies throughout the search. Furthermore, NNALNS outperforms ALNS on all of the test instances except for V2-d70-e6. This suggests that we can learn better selection strategies using RL than the selection strategies arising from the adaptive weight scheme in ALNS.

Table 8.7: Summary of results from the comparative study. The objective values and improvements are after 1000 iterations, averaged over 20 tests, and the best performing algorithm is emphasized in bold text.

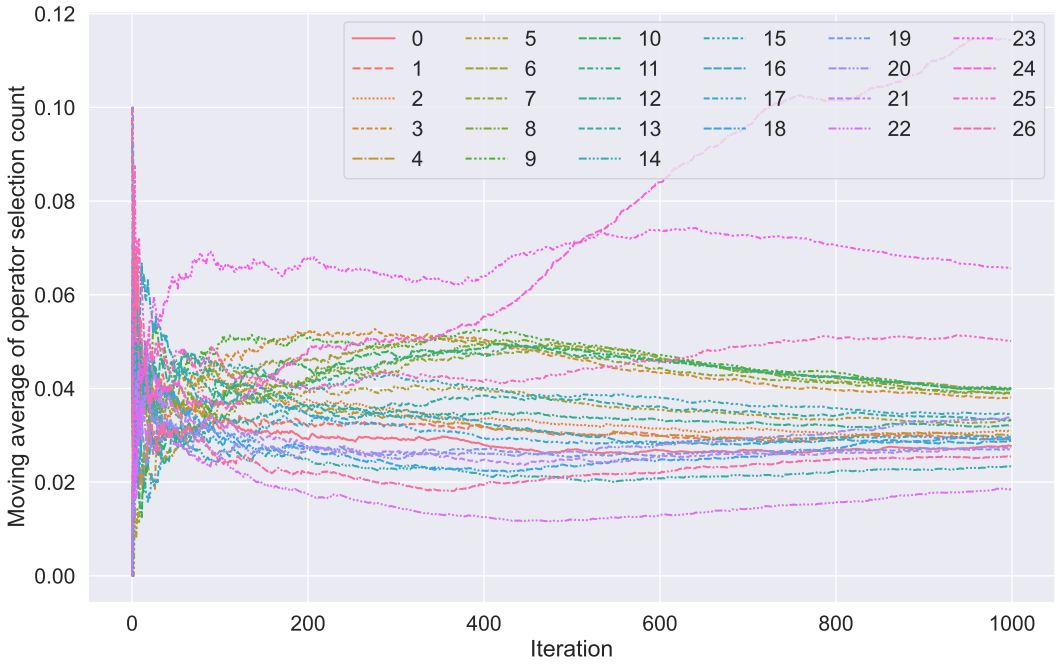
	ULNS	**ALNS**		**NNALNS**	
Instance	Obj. val	Impr.(%)	Obj. val.	Impr.(%)	Obj. val.
B1-d14-e14	-85.948	11.58	-95.900	**13.94**	-97.930
B2-d28-e16	-72.714	32.15	-96.094	**45.08**	-105.494
B3-d28-e50	11.947	533.59	-51.801	**799.17**	-83.530
B4-d42-e45	1351.447	4.94	1284.734	**15.60**	1140.622
B5-d56-e20	321.291	27.28	233.650	**57.60**	136.238
B6-d84-e22	423.651	64.25	151.467	**105.07**	-21.497
B7-d182-e50	2052.894	35.81	1317.684	**62.64**	766.906
V1-d56-e9	996.674	10.42	892.856	**13.57**	861.453
V2-d70-e6	-244.357	**10.18**	-269.229	4.57	-255.526
V3-d46-e28	-244.114	83.60	-448.195	**86.52**	-455.322
V4-d84-e8	787.358	7.31	729.753	**10.61**	703.825
V5-d42-e15	1015.204	18.43	828.073	**23.26**	779.038
V6-d98-e15	2619.869	17.85	2152.315	**28.53**	1872.520

8.5 Selection Strategies

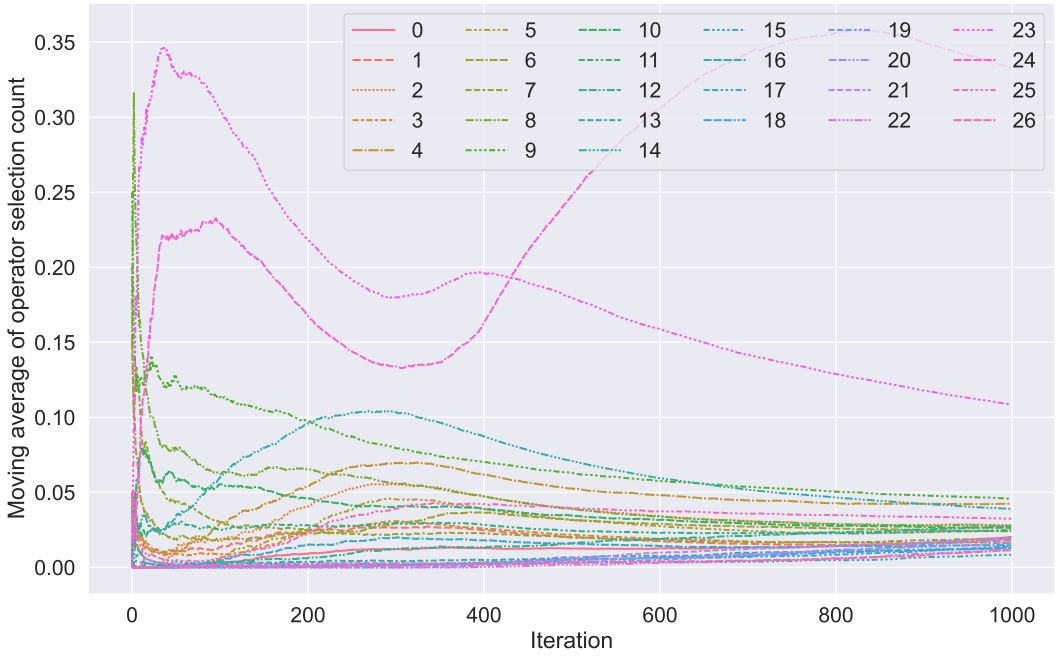
In this section, we take a closer look at the operator selection policies of NNALNS and how they compare to ALNS in terms of operator choices at different iteration stages. The goal is to provide insight into similarities and differences between ALNS and NNALNS in terms of selection strategies.

Figure 8.6 shows how operators are selected for each iteration on problem V5-d42-e15. It is a moving average of the number of times each operator is chosen. The figure clearly illustrates the key difference between ALNS and NNALNS. At the early iteration stages, ALNS selects operators based on a uniform probability distribution and gradually learns preferred operators based on the online feedback obtained during the search. NNALNS on the other hand, exploits information acquired during training already from the first iteration.

We observe that both ALNS and NNALNS tend to favor the same operators, which indicates that NNALNS manages to capture generally high-performing operators on the problem instance. This trend is present in the majority of the test instances. However, NNALNS seems to choose these operators more frequently than ALNS; notice the differences in values on the y-axis. Another interesting observation is that the plots show a correlation between the selection strategies and the iteration stage for both methods. Particularly, we see a shift in selection strategy after 400 iterations when the acceptance criterion only accepts improving solutions. Additional operator development plots are found in Appendix C.4



(a) ALNS



(b) NNALNS

Figure 8.6: Moving average of operator selection count for ALNS and NNALNS for V5-d42-e15. Notice the differences between the two plots in values on the y-axis.

Figure 8.7 and Figure 8.8 show the total number of times each operator was selected for all instances. See Table B.3 and Table B.4 for the mapping between operator number and name. We observe that for the benchmark instances, NNALNS tends to favor the large destroy-repair operators like operators 1 and 5. In contrast, for the Visma instances, the hybrid operators like operators 24 and 23 are selected more frequently. These observations indicate structural differences between the two datasets that induce different optimal selection policies. The differences can impede generalized learning between the two datasets.

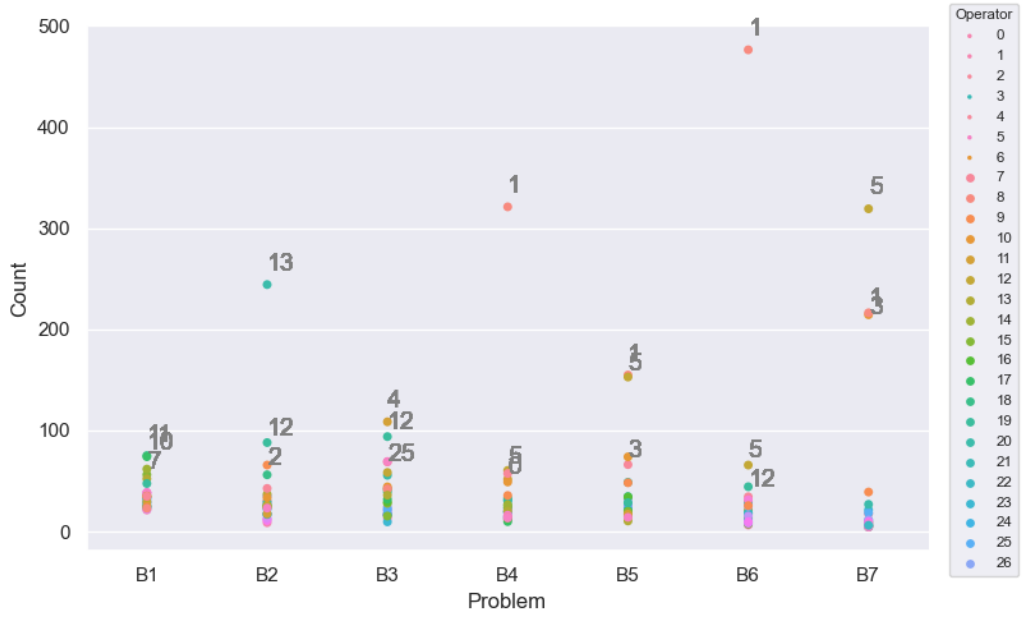


Figure 8.7: Total operator selection count for B1-B7. The three most frequently chosen operators are labeled in the figure.

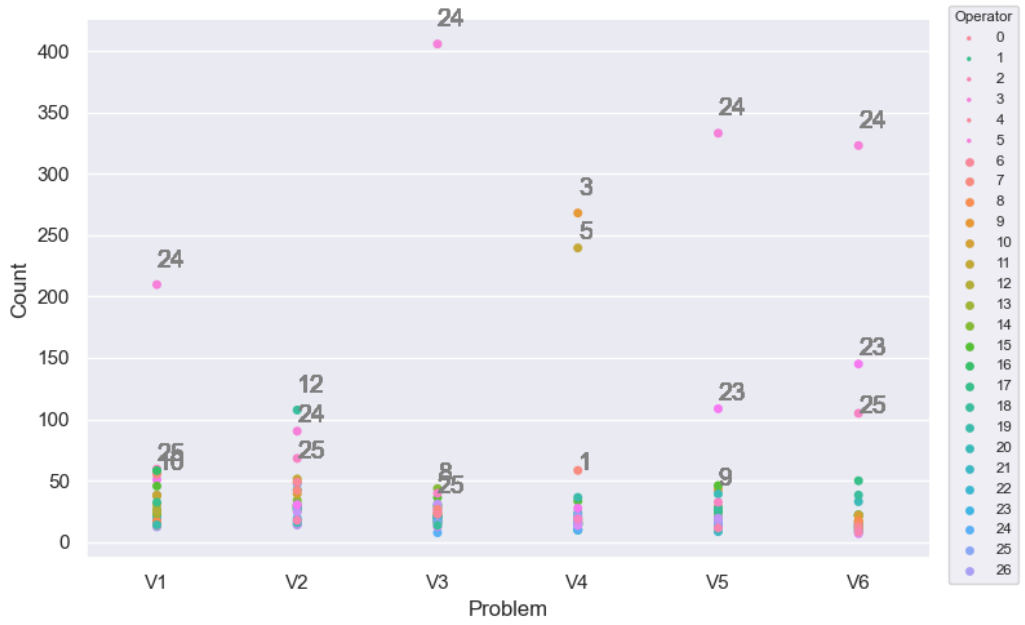


Figure 8.8: Total operator selection count for V1-V6. The three most frequently chosen operators are labeled in the figure.

8.6 Generalized Learning

The value of using a pre-trained operator selection policy as in NNALNS relies on its reusability across different variations of the problem. In this section, we investigate the generalizability of learned selection policies. Section 8.6.1 explores NNALNS' ability to learn selection policies between separate instances, whilst Section 8.6.2 measures its adaptability to small changes of the problem instances.

8.6.1 Cross-Instance Learning

A desirable property for NNALNS is to learn policies that are general across instances. To measure this property, we divide the problem instances into a training and test set as we explain in Section 8.2.

Table 8.8 summarizes the results when applying operator policies obtained from the training set on the test set. For all the benchmark instances in the test set, we observe that NNALNS outperforms ALNS regarding objective value, while for the Visma instances, this is not the case. This discrepancy suggests that although the benchmark instances vary more in size than the Visma instances, they are still more homogeneous. This might be due to the fact that the benchmark instances are created in a more standardized manner. The results indicate that a single optimal selection policy is hard to capture for real-world instances. Nevertheless, NNALNS still performs better than ULNS on all instances, implying that it is possible to attain policies applicable to unseen real-world instances. However, more refined parameter tuning might be needed, which is a topic for further research.

Table 8.8: Performance of NNALNS on test instance, when trained on train instances.

	ULNS	**ALNS**		**NNALNS**	
Instance	Obj. val	Impr.(%)	Obj. val	Impr.(%)	Obj. val
B5-d56-e20	318.221	25.40	237.388	**43.08**	181.139
B6-d84-e22	370.949	50.95	181.940	**69.06**	114.788
B7-d182-e50	2097.666	36.22	1337.945	**43.96**	1175.621
V1-d56-e9	1017.669	**13.66**	878.639	12.48	890.610
V3-d46-e28	-246.209	**83.12**	-450.864	75.15	-431.239
V5-d42-e15	968.22	**12.87**	843.631	12.69	845.394

8.6.2 Adaptability

As we see in Section 8.6.1, NNALNS struggles to learn a common selection policy for the real-world instances provided by Visma Resolve, when compared to ALNS. We now look at the effect of training NNALNS on small permutations of the same instance to measure its robustness to smaller changes to the instances. Such modifications reflect real-world cases where, for example, a new employee is hired or there are sudden changes to demand during or between scheduling periods. We limit this test to the Visma instances, as cross-instance learning was possible for the benchmark dataset. Furthermore, we regard adaptability to be most interesting to examine in the case of real-world instances.

Table 8.9: The performance of NNALNS on V1-V5 when trained on smaller modifications of the same problem instance.

	ULNS	**ALNS**		**NNALNS**	
Instance	Obj. val	Impr.(%)	Obj. val	Impr.(%)	Obj. val
V1-d56-e9	966.201	4.94	918.608	**15.96**	812.008
V2-d70-e6	-234.051	**14.08**	-266.996	6.274	-248.735
V3-d46-e28	-247.351	80.11	-445.503	**99.71**	-493.972
V4-d84-e8	762.968	3.02	739.937	**5.51**	720.968
V5-d42-e15	962.171	11.96	847.072	**12.85**	838.569

Table 8.9 shows the performance of NNALNS when the policy is trained on modifications of the Visma instances. In this experiment, NNALNS outperforms ULNS and ALNS on all problem instances except V2-d70-e6, indicating that the policy trained on modifications of the problem is reusable on the original problem instance. It is not surprising that the result for adaptability is not favorable for V2-d70-e6, as NNALNS does not outperform ALNS on this instance in the comparative study.

In a real-world use case, these results points in favor of using a pre-trained operator selection policy. The RRP for a particular customer of Visma Resolve is often similar in structure between scheduling periods. By storing a pre-trained policy for a particular customer, information acquired from previous periods can be reused for

a new scheduling period to obtain lower objective values without having to train a new policy.

8.7 Limitations

There are some limitations to how we conducted this computational study. First of all, there is considerable stochasticity in our solution method. Thus, the result of training sessions can vary and yield different policies. However, we have only trained NNALNS two times per instance or set of instances, once with and without problem-specific features. We performed 20 tests per policy, which could have been increased if not for limited computational capacity.

Another limitation is that the policy selection, as described in Section 8.2, is done through a somewhat subjective process. Due to the stochasticity during training, we do an evaluation of policies that have proved more effective than simply selecting the final one. Such policy selection can present an issue of consistency, as it requires insight into the algorithm and is inherently subjective. Furthermore, there might be superior policies that we have not tested. Developing a more robust policy selection method could be in order, both for consistency and better results.

Additionally, time is an aspect not considered in this computational study. The goal of this chapter is to evaluate the performance of NNALNS, and thus the potential of such a solution method. Training times are therefore not relevant for the discussion. Furthermore, the results in Section 8.6 indicate that one can use pre-trained policies on new problem instances, which lessens the cost of training. Solution times are another matter, but as the time to obtain an output from the ANN in NNALNS is negligible, the solution times of NNALNS and ALNS are equivalent. Potential differences in time are rather due to operator selection.

Finally, there is a large number of hyperparameters, particularly related to the RL training algorithm, which we have not tuned rigorously. We consider such tuning out-of-scope due to time restrictions. More appropriate hyperparameter values could yield better results, and extensive hyperparameter testing would provide further insight into the solution method.

Chapter 9

Concluding Remarks and Future Research

We have set four overarching goals for this thesis; we want to investigate whether NNALNS can learn more intelligent operator selection strategies for the RRP compared to ALNS, include an enriched state representation of an RRP solution for NNALNS, examine the generalizability of learned selection strategies across different RRP instances, and develop operators tailored to the RRP targeting different properties of a solution.

Section 9.1 concludes the most important results and main contributions of this master's thesis. Following, Section 9.2 discusses interesting directions for future research.

9.1 Concluding Remarks

ML is a field with many recent advances that is currently of great interest. Visma Resolve has expressed interest in integrating ML methods in traditional OR solution methods to use for their PRP. We name this specific PRP the RRP. In this thesis, we have presented NNALNS, which is a modification of ALNS that integrates ML by using an ANN that parameterizes the operator selection policy. We have implemented many operators that target different aspects of an RRP solution. We train the ANN using a training algorithm from the state-of-the-art family of RL algorithms, PPO. The ANN uses a feature vector representing the state as input to output a probability distribution over the operators, and we have explored an enriched state representation that includes both search-based and problem-specific features in the feature vector.

The results of the computational study show that the operators we include successfully improve the objective value from an initial solution. In fact, even a version of ALNS with a constant, uniform probability distribution over the operators, namely ULNS, can significantly improve the objective value. Furthermore, the results show that ALNS and NNALNS, with varying probability distributions over the operators

depending on the iteration stage, can take this improvement even further.

NNALNS outperforms ALNS on all instances but one in terms of objective value when the policy is trained on one instance and tested on the same. We also see from the results that there is a difference between the operator selection strategies of NNALNS and ALNS. However, they are similar in the sense that the most frequently selected operators for NNALNS are often the most selected operators for ALNS. Both of these observations indicate that NNALNS is able to learn better operator selection strategies than those that arise from ALNS.

In addition, we have looked at the performance of NNALNS in terms of generalizability. The results vary when we train the policy on a subset of the instances and test on a different subset, which we refer to as cross-instance learning. The results suggest that NNALNS with cross-instance policies can outperform ALNS when the training and test sets consist of benchmark instances. However, the results do not show that such policies can be used successfully with NNALNS on the real-world instances provided by Visma Resolve.

We have also looked at generalizability in terms of adaptability. Adaptability is when the policy is trained on several permutations of an RRP instance and tested on the original. The permutations represent realistic changes to an RRP instance for a particular organization from one scheduling period to the next. The results for the real-world Visma instances are promising for adaptability; on all instances but one, NNALNS outperforms ALNS. For a real-world use case, the results point in favor of using a pre-trained operator selection policy. The RRP for a particular customer of Visma Resolve is often similar in structure between scheduling periods. By storing a pre-trained policy for a particular customer, information acquired from previous periods can be reused for a new scheduling period to obtain lower objective values without training a new policy.

The results of the computational study show that NNALNS with a policy that uses only search-based features outperforms NNALNS with a policy that uses search-based and problem-specific features in terms of objective value on most problem instances. On the contrary, NNALNS with all features outperforms NNALNS with only search-based features in terms of cumulative reward on most instances. This points toward reward hacking; the addition of problem-specific features allows the policy to exploit structures of an instance to obtain more rewards at the expense of objective value. Another possible explanation is simply that more features require more time in training.

In conclusion, the results show that NNALNS is a promising solution approach to the RRP. Especially, the results regarding adaptivity on the Visma instances have significant implications for real-world use cases. Using pre-trained networks, an RL agent can learn to exploit the structures for a particular client of Visma Resolve, thus performing well on reasonable changes to the problem.

9.2 Future Research

One important goal of future research would be to address the potential reward hacking described in Section 8.3.2. Applying a different reward function could help close the gap between the reward- and objective functions and align the training with the problem definition, thus improving the performance of NNALNS. It could also allow NNALNS to benefit from an enriched state representation, as a problem-specific feature vector has shown to yield larger rewards. Ideas to explore are a more sparse reward scheme or further integrating the objective function into the reward function. Section 8.3.1 and Section 8.3.2 also discuss that more time in training might be required when we use both the search-based features and problem-specific features. Therefore, this should also be a topic for future research as it could potentially increase the performance of NNALNS with the enriched state representation.

Future research into different unexplored aspects of the solution method could also improve its performance. Rigorous hyperparameter tuning could yield more effective values where we have opted to use standard ones. Another possibility for improvement would be to test different acceptance criteria, which may guide the search more advantageously than those tested. Similarly, the problem-specific features and operators have remained the same throughout testing, and more rigorous testing of other selections could yield choices that would improve performance. A final way to improve NNALNS could be to create an algorithm for selecting policies. Such an algorithm could return better policies than doing this subjectively and potentially stop the training when a certain stopping criterion is met to save time.

Finally, one could explore other ways to integrate RL into ALNS. A possible alteration of NNALNS would be to combine the offline learning with the online adaptability of ALNS by scaling the probability distribution from the ANN according to the adaptive weights in ALNS. Another application area could be the acceptance criterion, where an ANN could parameterize the acceptance criterion instead of the operator selection strategy.

Bibliography

Abobaker, R. A., Ayob, M. and Hadwan, M. (2011), Greedy constructive heuristic and local search algorithm for solving nurse rostering problems, *in* '2011 3rd Conference on Data Mining and Optimization (DMO)', IEEE, pp. 194–198.

Aickelin, U. and Dowsland, K. A. (2004), 'An indirect genetic algorithm for a nurse-scheduling problem', *Computers & operations research* **31**(5), 761–778.

Alvarez, A. M., Louveaux, Q. and Wehenkel, L. (2014), A supervised machine learning approach to variable branching in branch-and-bound, *in* 'In ecml', Citeseer.

Anwar, K., Awadallah, M. A., Khader, A. T. and Al-betar, M. A. (2014), Hyper-heuristic approach for solving nurse rostering problem, *in* '2014 IEEE Symposium on Computational Intelligence in Ensemble Learning (CIEL)', pp. 1–6.

Arulkumaran, K., Deisenroth, M. P., Brundage, M. and Bharath, A. A. (2017), 'Deep reinforcement learning: A brief survey', *IEEE Signal Processing Magazine* **34**(6), 26–38.

Asta, S., Özcan, E. and Curtois, T. (2016), 'A tensor based hyper-heuristic for nurse rostering', *Knowledge-based systems* **98**, 185–199.

Bai, R., Burke, E. K., Kendall, G., Li, J. and McCollum, B. (2010), 'A hybrid evolutionary approach to the nurse rostering problem', *IEEE Transactions on Evolutionary Computation* **14**(4), 580–590.

Balcan, M.-F., Dick, T., Sandholm, T. and Vitercik, E. (2018), Learning to branch, *in* J. Dy and A. Krause, eds, 'Proceedings of the 35th International Conference on Machine Learning', Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 344–353.

Bard, J. F. and Purnomo, H. W. (2005), 'Preference scheduling for nurses using column generation', *European Journal of Operational Research* **164**(2), 510–534.

Beckmann, F. R. and Klyve, K. K. (2016), Optimisation-based nurse scheduling for real-life instances, Master's thesis, NTNU.

Beddoe, G., Petrovic, S. and Li, J. (2009), 'A hybrid metaheuristic case-based reasoning system for nurse rostering', *Journal of Scheduling* **12**(2), 99–119.

Bellanti, F., Carello, G., Della Croce, F. and Tadei, R. (2004), 'A greedy-based neighborhood search approach to a nurse rostering problem', *European Journal of Operational Research* **153**(1), 28–40.

Bello, I., Pham, H., Le, Q. V., Norouzi, M. and Bengio, S. (2016), 'Neural combinatorial optimization with reinforcement learning', *CoRR* .

Bengio, Y., Lodi, A. and Prouvost, A. (2021), 'Machine learning for combinatorial optimization: A methodological tour d'horizon', *European Journal of Operational Research* **290**(2), 405–421.

Bilgin, B. (2012), Advanced Models and Solution Methods for Automation of Personnel Rostering Optimisation, PhD thesis, Katholieke Universiteit Leuven.

Brucker, P., Burke, E. K., Curtois, T., Qu, R. and Vanden Berghe, G. (2010), 'A shift sequence based approach for nurse scheduling and a new benchmark dataset', *Journal of Heuristics* **16**(4), 559–573.

Brucker, P., Qu, R., Burke, E. and Post, G. (2005), A decomposition, construction and post-processing approach for nurse rostering, New York.

Bunton, J. D., Ernst, A. T. and Krishnamoorthy, M. (2017), An integer programming based ant colony optimisation method for nurse rostering, *in* '2017 Federated Conference on Computer Science and Information Systems (FedCSIS)', IEEE, pp. 407–414.

Burke, E., De Causmaecker, P., Petrovic, S. and Berghe, G. V. (2003), Variable neighborhood search for nurse rostering problems, *in* 'Metaheuristics: computer decision-making', Springer, pp. 153–172.

Burke, E. K., Causmaecker, P. D., Petrovic, S. and Berghe, G. V. (2006), 'Metaheuristics for handling time interval coverage constraints in nurse scheduling', *Applied Artificial Intelligence* **20**(9), 743–766.

Burke, E. K., Curtois, T., Qu, R. and Vanden-Berghe, G. (2008), 'Problem model for nurse rostering benchmark instances', *ASAP, School of Computer Science, University of Nottingham, Jubilee Campus, Nottingham, UK* .

Burke, E. K., De Causmaecker, P., Berghe, G. V. and Van Landeghem, H. (2004), 'The state of the art of nurse rostering', *Journal of scheduling* **7**(6), 441–499.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E. and Woodward, J. R. (2019), A classification of hyper-heuristic approaches: revisited, *in* 'Handbook of metaheuristics', Springer, pp. 453–477.

Burke, E. K., Kendall, G. and Soubeiga, E. (2003), 'A tabu-search hyperheuristic for timetabling and rostering', *Journal of heuristics* **9**(6), 451–470.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S. (2003), Hyper-heuristics: An emerging direction in modern search technology, *in* 'Handbook of metaheuristics', Springer, pp. 457–474.

Ceschia, S., Dang, N., De Causmaecker, P., Haspeslagh, S. and Schaerf, A. (2019), 'The second international nurse rostering competition', *Annals of Operations Research* **274**(1), 171–186.

Ceschia, S., Guido, R. and Schaerf, A. (2020), 'Solving the static inrc-ii nurse rostering problem by simulated annealing based on large neighborhoods', *Annals of Operations Research* **288**(1), 95–113.

Chen, Z., De Causmaecker, P. and Dou, Y. (2021), Deep neural networked assisted tree search for the personnel rostering problem, *in* 'Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT', Vol. 2.

Chen, Z., De Causmaecker, P. and Dou, Y. (2022), 'A combined mixed integer programming and deep neural network-assisted heuristics algorithm for the nurse rostering problem', *Available at SSRN 4020057* .

Cowling, P., Kendall, G. and Soubeiga, E. (2001), A hyperheuristic approach to scheduling a sales summit, *in* E. Burke and W. Erben, eds, 'Practice and Theory of Automated Timetabling III', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 176–190.

Cybenko, G. V. (1989), 'Approximation by superpositions of a sigmoidal function', *Mathematics of Control, Signals and Systems* **2**, 303–314.

Dagbladet (2021), 'Får coronakjeft: - unødig'. [Online; accessed November 15, 2021]. **URL:** *https://www.dagbladet.no/nyheter/far-coronakjeft—unodig/74631099*

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B. and Song, L. (2017), 'Learning combinatorial optimization algorithms over graphs', *CoRR* .

Dean, J. (2008), Staff scheduling by a genetic algorithm with a two-dimensional chromosome structure, *in* 'Proc of the 7th Conference on the Practice and Theory of Automated Timetabling, Montreal, Canada'.

Della Croce, F. and Salassa, F. (2014), 'A variable neighborhood search based matheuristic for nurse rostering problems', *Annals of Operations Research* **218**, 1–15.

Derrow-Pinion, A., She, J., Wong, D., Lange, O., Hester, T., Perez, L., Nunkesser, M., Lee, S., Guo, X., Wiltshire, B., Battaglia, P. W., Gupta, V., Li, A., Xu, Z., Sanchez-Gonzalez, A., Li, Y. and Velickovic, P. (2021), *ETA Prediction with Graph Neural Networks in Google Maps*, Association for Computing Machinery, New York, NY, USA, p. 3767–3776.

Dowsland, K. A. (1998), 'Nurse scheduling with tabu search and strategic oscillation', *European journal of operational research* **106**(2-3), 393–407.

Dueck, G. and Scheuer, T. (1990), 'Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing', *Journal of Computational Physics* **90**(1), 161–175.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B. and Sier, D. (2004), 'An annotated bibliography of personnel scheduling and rostering', *Annals of Operations Research* **127**(1), 21–144.

Ernst, A. T., Jiang, H., Krishnamoorthy, M. and Sier, D. (2004), 'Staff scheduling and rostering: A review of applications, methods and models', *European journal of operational research* **153**(1), 3–27.

Forsyth, P. and Wren, A. (1997), 'An ant system for bus driver scheduling', *Research Report Series - University of Leeds School of Computer Science* .

Gendreau, M. and Potvin, J.-Y. (2010), *Handbook of Metaheuristics*, Springer New York, NY.

Gregory, P. (2010), 'Nurse rostering competition entry: Lns, restarts and state machines', *Annals of Operations Research* .

Grov, H., Kallevik, E. and Nærland, S. (2020), Optimizing real-life personnel scheduling problems through shift design and a parallel adaptive large neighborhood search, Master's thesis, NTNU.

Gu, S. and Hao, T. (2018), A pointer network based deep learning algorithm for 0–1 knapsack problem, *in* '2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)', pp. 473–477.

Guessoum, F., Haddadi, S. and Gattal, E. (2020), 'Simple, yet fast and effective two-phase method for nurse rostering', *American Journal of Mathematical and Management Sciences* **39**(1), 1–19.

Hadwan, M. and Ayob, M. (2010), A constructive shift patterns approach with simulated annealing for nurse rostering problem, *in* '2010 International Symposium on Information Technology', Vol. 1, IEEE, pp. 1–6.

Hans, E. W., Van Houdenhoven, M. and Hulshof, P. J. (2012), A framework for healthcare planning and control, *in* 'Handbook of healthcare system scheduling', Springer, pp. 303–320.

Jordan, M. I. and Mitchell, T. M. (2015), 'Machine learning: Trends, perspectives, and prospects', *Science* **349**(6245), 255–260.

Kallestad, J. V. (2021), Developing an intelligent hyperheuristic for combinatorial optimization problems using deep reinforcement learning, Master's thesis, The University of Bergen.

Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M. and Talbi, E.-G. (2022), 'Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art', *European Journal of Operational Research* **296**(2), 393–422.

Khalil, E., Le Bodic, P., Song, L., Nemhauser, G. and Dilkina, B. (2016), Learning to branch in mixed integer programming, *in* 'Proceedings of the AAAI Conference on Artificial Intelligence', Vol. 30.

Kiefer, A. (2015), Large Neighborhood Search for the Nurse Rostering Problem, PhD thesis, Technische Universität Wien.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983), 'Optimization by simulated annealing', *Science* **220**(4598), 671–680.

KNIME (2021), 'A friendly introduction to [deep] neural networks'. [Online; accessed May 5, 2022].
URL: *https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks*

Kruber, M., Lübbecke, M. E. and Parmentier, A. (2017), Learning when to use a decomposition, *in* 'International conference on AI and OR techniques in constraint programming for combinatorial optimization problems', Springer, pp. 202–210.

Kumar, M., Teso, S., De Causmaecker, P. and De Raedt, L. (2019), Automating personnel rostering by learning constraints using tensors, *in* '2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)', pp. 697–704.

Langfeldt, S. S., Spangelo, T. H. and Langholm, M. H. (2021), A stepping horizon construction heuristic for personnel rostering.

Lundgren, J., Rönnqvist, M. and Värbrand, P. (2008), *Optimization*, Studentlitteratur.

Maenhout, B. and Vanhoucke, M. (2010), 'Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem', *Journal of scheduling* **13**(1), 77–93.

Mehta, H., Kanani, P. and Lande, P. (2019), 'Google maps', *Int. J. Comput. Appl* **178**(8), 41–46.

Messelis, T. and De Causmaecker, P. (2010), 'An nrp feature set'.

Messelis, T., Haspeslagh, S., Bilgin, B., De Causmaecker, P. and Vanden Berghe, G. (2009), Towards prediction of algorithm performance in real world optimisation problems, *in* 'Proceedings of the 21st Benelux conference on Artificial Intelligence.', pp. 177–183.

Mischek, F. and Musliu, N. (2016), Integer programming and heuristic approaches for a multi-stage nurse rostering problem, *in* 'PATAT 2016: Proceedings of the 11th international conference of the practice and theory of automated timetabling, PATAT'.

Mitchell, T. (1998), *Machine Learning*, McGraw Hill, Ohio.

Nair, V., Alizadeh, M. et al. (2020), Neural large neighborhood search, *in* 'Learning Meets Combinatorial Algorithms at NeurIPS2020'.

Norweginan Working Environments Act §10-4 (2015).

Nurmi, K., Kyngäs, J. and Kyngäs, N. (2016), 'Synthesis of employer and employee satisfaction–case nurse rostering in a finnish hospital', *Journal of Advances in Information Technology Vol* **7**(2).

Nurse Rostering Benchmark Instances (2014). [Online; accessed January 30, 2022]. **URL:** *http://www.schedulingbenchmarks.org/nrp/*

Oberweger, F. F. (2021), A Learning Large Neighborhood Search for the Staff Rerostering Problem, PhD thesis, Wien.

Ouelhadj, D., Martin, S., Smet, P., Ozcan, E. and Berghe, G. V. (2012), 'Fairness in nurse rostering'.

Pato, M. V. and Moz, M. (2008), 'Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic', *Journal of Heuristics* **14**(4), 359–374.

Pillay, N. and Qu, R. (2019), *Hyper-Heuristics: Theory and applications*, Natural Computing Series, Springer Nature, Cham, Switzerland.

Pisinger, D. and Ropke, S. (2010), Large neighborhood search, *in* 'Handbook of metaheuristics', Springer, pp. 399–419.

Ramli, R., Ahmad, S. N. I., Abdul-Rahman, S. and Wibowo, A. (2020), 'A tabu search approach with embedded nurse preferences for solving nurse rostering problem', *International Journal for Simulation and Multidisciplinary Design Optimization* **11**, 10.

Range, T. M. (2021), Lecture on hospital optimization, IØT NTNU.

Rönnberg, E., Larsson, T. and Bertilsson, A. (2013), Automatic scheduling of nurses: What does it take in practice?, *in* 'Systems Analysis Tools for Better Health Care Delivery', Springer, pp. 151–178.

Ropke, S. and Pisinger, D. (2006), 'An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows', *Transportation science* **40**(4), 455–472.

Ross, P. and Marfn-Blazquez, J. (2005), Constructive hyper-heuristics in class timetabling, *in* '2005 IEEE Congress on Evolutionary Computation', Vol. 2, IEEE, pp. 1493–1500.

Sabharwal, A., Samulowitz, H. and Reddy, C. (2012), Guiding combinatorial optimization with uct, *in* 'International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming', Springer, pp. 356–361.

Schrack, J., Ortega, R., Dabu, K., Truong, D., Aibin, M. and Aibin, A. (2021), Combining tabu search and genetic algorithm to determine optimal nurse schedules, *in* '2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)', pp. 1–7.

Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P. (2015), 'High-dimensional continuous control using generalized advantage estimation'.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017), 'Proximal policy optimization algorithms.', *CoRR* .

Shaw, P. (1998), Using constraint programming and local search methods to solve vehicle routing problems, *in* 'International conference on principles and practice of constraint programming', Springer, pp. 417–431.

Shi, P. and Landa-Silva, D. (2018), Approximate dynamic programming with combined policy functions for solving multi-stage nurse rostering problem, *in* G. Nicosia, P. Pardalos, G. Giuffrida and R. Umeton, eds, 'Machine Learning, Optimization, and Big Data', Springer International Publishing, Cham, pp. 349–361.

Smet, P. (2018), Constraint reformulation for nurse rostering problems, *in* 'Proceedings of the 12th international conference on the practice and theory of automated timetabling', PATAT, pp. 69–80.

Smet, P., Ernst, A. T. and Berghe, G. V. (2016), 'Heuristic decomposition approaches for an integrated task scheduling and personnel rostering problem', *Computers & Operations Research* **76**, 60–72.

Smet, P. and Vanden Berghe, G. (2012), A matheuristic approach to the shift minimisation personnel task scheduling problem, *in* '9th International Conference on the Practice and Theory of Automated Timetabling', Citeseer, pp. 145–160.

Smet, P. and Vanden Berghe, G. (2016), Large neighbourhood search for large-scale shift assignment problems with multiple tasks, pp. 339–352.

SSB (2013), 'Fra jordbruk til tjenester'. [Online; accessed November 21, 2021].
URL: *https://www.ssb.no/nasjonalregnskap-og-konjunkturer/artikler-og-publikasjoner/_attachment/152574?_ts=142c712cb58*

SSB (2019), 'Eldrebølgen legger press på flere omsorgstjenester i kommunen'. [Online; accessed November 20, 2021].
URL: *https://www.ssb.no/helse/artikler-og-publikasjoner/eldrebolgen-legger-press-pa-flere-omsorgstjenester-i-kommunen*

SSB (2020), 'Fakta om norsk næringsliv'. [Online; accessed November 21, 2021].
URL: *https://www.ssb.no/nasjonalregnskap-og-konjunkturer/faktaside/norsk-naeringsliv*

Sutton, R. S. and Barto, A. G. (1998), *Reinforcement Learning: An Introduction*, MIT Press.

Sutton, R. S. and Barto, A. G. (2018), *Reinforcement Learning*, Adaptive Computation and Machine Learning series, 2 edn, Bradford Books, Cambridge, MA.

Syed, A. A., Akhnoukh, K., Kaltenhaeuser, B. and Bogenberger, K. (2019), Neural network based large neighborhood search algorithm for ride hailing services, *in* P. Moura Oliveira, P. Novais and L. P. Reis, eds, 'Progress in Artificial Intelligence', Springer International Publishing, Cham, pp. 584–595.

Thornton, J. and Sattar, A. (1997), Nurse rostering and integer programming revisited, *in* 'International conference on computational intelligence and multimedia applications', Citeseer, pp. 49–58.

Trilling, L., Guinet, A. and Le Magny, D. (2006), 'Nurse scheduling using integer linear programming and constraint programming', *IFAC Proceedings Volumes* **39**, 671–676.

Turhan, A. and Bilgen, B. (2020), 'A hybrid fix-and-optimize and simulated annealing approaches for nurse rostering problem', *Computers Industrial Engineering* **145**, 106531.

Turkeš, R., Sörensen, K. and Hvattum, L. M. (2021), 'Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search', *European Journal of Operational Research* **292**(2), 423–442.

Václavík, R., Ščha, P. and Hanzálek, Z. (2016), 'Roster evaluation based on classifiers for the nurse rostering problem', *Journal of Heuristics* **22**(5), 667–697.

Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E. and De Boeck, L. (2013), 'Personnel scheduling: A literature review', *European Journal of Operational Research* **226**(3), 367–385.

Wang, Y. (2010), 'A sociopsychological perspective on collective intelligence in metaheuristic computing', *International Journal of Applied Metaheuristic Computing* **1**(1), 110–128.

Wolbeck, L. A. (2019), 'Fairness aspects in personnel scheduling'.

Wu, J.-j., Lin, Y., Zhan, Z.-h., Chen, W.-n., Lin, Y.-b. and Chen, J.-y. (2013), An ant colony optimization approach for nurse rostering problem, *in* '2013 IEEE International Conference on Systems, Man, and Cybernetics', IEEE, pp. 1672–1676.

Yegnanarayana, B. (2009), *Artificial neural networks*, PHI Learning Pvt. Ltd.

Yuan, Y., Yu, Z. L., Gu, Z., Deng, X. and Li, Y. (2019), 'A novel multi-step reinforcement learning method for solving reward hacking', *Applied Intelligence* **49**(8), 2874–2888.

Zarpellon, G., Jo, J., Lodi, A. and Bengio, Y. (2020), 'Parameterizing branch-and-bound search trees to learn branching policies', *arXiv preprint arXiv:2002.05120* **12**.

Zhang, H. and Lu, J. (2008), 'Adaptive evolutionary programming based on reinforcement learning', *Information Sciences* **178**(4), 971–984.

Zhang, Y., Bai, R., Qu, R., Tu, C. and Jin, J. (2022), 'A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties', *European Journal of Operational Research* **300**(2), 418–427.

Zheng, A. and Casari, A. (2018), *Feature engineering for machine learning: principles and techniques for data scientists*, O'Reilly Media, Inc.

Appendix A

Mathematical Model

In this appendix we present a compact version of the mathematical model representing the RRP in Chapter 5.

A.1 Indices

e : Employee e
c : Competence c.
t : Time interval t.
s : Shift s.
r : Off shift r.
i : Day i.
j : Week j.

A.2 Sets

$\mathcal{I}$: Set of days.
$\mathcal{I}^{SAT}$: Set of Saturdays, excluding the final day in the planning horizon.
$\mathcal{J}$: Set of weeks.
$\mathcal{E}$: Set of all employees.
$\mathcal{E}_c^C$: Set of all employees with competence c.
$\mathcal{C}$: Set of competences.
$\mathcal{C}_e^E$: Set of the competences of employee e.
$\mathcal{T}$: Set of time intervals with demand.
$\mathcal{T}_r^O$: Set of time intervals overlapping weekly off shift r.
$\mathcal{S}$: Set of shifts.
$\mathcal{S}_i^I$: Set of shifts on day i.
$\mathcal{S}_t^O$: Set of shifts overlapping time interval t.
$\mathcal{S}_e^{VAL}$: Set of valid shifts for employee e, $\mathcal{S}_e^{VAL} \subseteq \mathcal{S}$.
$\mathcal{S}_{es}^{DRC}$: Set of shifts that breaks the daily rest rule for employee e if taken together with shift s, $\mathcal{S}_{es}^{DRC} \subseteq \mathcal{S}_e^{VAL}$.
$\mathcal{S}_{es}^{DRS}$: Set of shifts that breaks the daily rest rule for employee e if employee e is assigned to shift s and two of the shifts in $\mathcal{S}_{es}^{DRS}$, $\mathcal{S}_{es}^{DRS} \subseteq \mathcal{S}_e^{VAL}$.
$\mathcal{R}$: Set of weekly off shifts.
$\mathcal{R}_j$: Set of off shifts in week j.

A.3 Parameters

Instance parameters

$\underline{D}_{tc}$: Minimum demand coverage for competence c in time interval t.
D_{tc} : Ideal demand coverage for competence c in time interval t.
$\overline{D}_{tc}$: Maximum demand coverage for competence c in time interval t.
V_r^R : Duration of off shift r.
$\underline{V}^R$: Minimum weekly off shift duration.
$\overline{V}^R$: Maximum rewarded weekly off shift duration.
B_e : Weekly contracted hours for employee e.
L^{CD} : Desired maximum number of consecutive days.
P_{es} : Preference score for employee e working shift s.
V_s^S : Length of shift s.
$\overline{V}^S$: Desired shift length, upper limit.
$\underline{V}^S$: Desired shift length, lower limit.
V_t^T : Duration of time interval t.

Weights

A^F : Adjustment factor for weighting the lowest individual fairness cost.
A_e^B : Adjustment factor for weighting cost of deviating from contracted hours.
W^F : Weight of the lowest individual fairness cost.
W^{D+} : Weight of excess demand coverage.
W^{D-} : Weight of deficit demand coverage.
W^R : Weight of weekly off shift score.
W^B : Weight of contracted hours difference.
W^{PW} : Weight of partial weekends.
W^{IW} : Weight of isolated work days.
W^{IO} : Weight of isolated off days.
W^{CD} : Weight of consecutive days.
W^P : Weight of preference score.
W^{SL} : Weight of shift duration.

A.4 Decision variables

$$x_{es} = \begin{cases} 1, & \text{if employee } e \text{ is assigned to shift } s \\ 0, & \text{otherwise} \end{cases}$$

$$y_{etc} = \begin{cases} 1, & \text{if employee } e \text{ covers one unit of demand in time interval } t \\ & \text{with competence } c. \\ 0, & \text{otherwise} \end{cases}$$

$$w_{er} = \begin{cases} 1, & \text{if employee } e \text{ takes the off shift } r \\ 0, & \text{otherwise} \end{cases}$$

$$\gamma_{ei} = \begin{cases} 1, & \text{if employee } e \text{ takes a shift on day } i \\ 0, & \text{otherwise} \end{cases}$$

$$\rho_{ei}^{SAT}, \rho_{ei}^{SUN} = \begin{cases} 1, & \text{if employee } e \text{ works a partial weekend by working} \\ & \text{Saturday or Sunday, respectively} \\ 0, & \text{otherwise} \end{cases}$$

$$\sigma_{ei} = \begin{cases} 1, & \text{if employee } e \text{ works an isolated day on day } i \\ 0, & \text{otherwise} \end{cases}$$

$$\phi_{ei} = \begin{cases} 1, & \text{if employee } e \text{ takes an isolated off day on day } i \\ 0, & \text{otherwise} \end{cases}$$

$$\pi_{ei} = \begin{cases} 1, & \text{if employee } e \text{ starts a sequence of consecutive working days} \\ & \text{exceeding the desired limit on day } i \\ 0, & \text{otherwise} \end{cases}$$

$\delta_{tc}^{+}, \delta_{tc}^{-}$ = respectively excess and deficit demand coverage with respect to ideal. demand in time interval t for competence c.

f_e = variable for storing fairness costs for employee e.

g = variable for storing the highest fairness cost.

$\lambda_e^{+}, \lambda_e^{-}$ = deviation between worked and contracted hours for employee e.

μ_{tc} = difference between covered and minimum demand for competence c in time interval t.

A.5 Objective function

$$\min z = \sum_{e \in \mathcal{E}} f_e \tag{A.1a}$$

$$+ W^F A^F g \tag{A.1b}$$

$$\sum_{t \in \mathcal{T}} V_t^T \sum_{c \in \mathcal{C}} (W^{D+} \delta_{tc}^{+} + W^{D-} \delta_{tc}^{-}) \tag{A.1c}$$

A.6 Constraints

Demand constraints

$$\sum_{e \in \mathcal{E}_c^C} y_{etc} = \underline{D}_{tc} + \mu_{tc} \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{A.2}$$

$$\mu_{tc} \leq \overline{D}_{tc} - \underline{D}_{tc} \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{A.3}$$

$$\mu_{tc} + \underline{D}_{tc} - D_{tc} = \delta_{tc}^{+} - \delta_{tc}^{-} \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{A.4}$$

Work allocation

$$\sum_{s \in \mathcal{S}_i^I} x_{es} = \gamma_{ei} \qquad \forall e \in \mathcal{E}, i \in \mathcal{I} \tag{A.5}$$

$$\sum_{s \in \mathcal{S}_t} x_{es} = \sum_{c \in \mathcal{C}} y_{etc} \qquad \forall e \in \mathcal{E}, t \in \mathcal{T} \tag{A.6}$$

$$\sum_{c \in \mathcal{C}} y_{etc} \leq 1 \qquad \forall e \in \mathcal{E}, t \in \mathcal{T} \tag{A.7}$$

Rest constraints

$$2x_{es} + \sum_{s' \in S_{es}^{DRC}} x_{es'} \leq 2 \qquad \forall e \in \mathcal{E}, s \in \mathcal{S} \tag{A.8}$$

$$x_{es} + \sum_{s' \in S_{es}^{DRS}} x_{es'} \leq 2 \qquad \forall e \in \mathcal{E}, s \in \mathcal{S} \tag{A.9}$$

$$\sum_{r \in R_j^J} w_{er} = 1 \qquad \forall e \in \mathcal{E}, j \in \mathcal{J} \tag{A.10}$$

$$|\mathcal{T}_r^R| w_{er} \leq \sum_{t \in \mathcal{T}_r^R} (1 - \sum_{c \in C} y_{etc}) \qquad \forall e \in \mathcal{E}, r \in \mathcal{R} \tag{A.11}$$

Fairness

$$f_e = W^{IW} \sum_{i \in I} \sigma_i \tag{A.12a}$$

$$+ W_{IO} \sum_{i \in I} \phi_{ei} \tag{A.12b}$$

$$+ W_{CD} \sum_{i \in I} \pi_{ei} \tag{A.12c}$$

$$+ W_{PW} \sum_{i \in I^{SAT}} (\rho^{SAT}_{ei} + \rho^{SUN}_{e(i+1)}) \tag{A.12d}$$

$$+ W_{SL} \sum_{s \in S_e^{VAL}} \max\left(V^S_s - \overline{V}^S, \underline{V}^S - V^S_s, 0\right) x_{es} \tag{A.12e}$$

$$+ W_B A^B_e (\lambda^+_e + \lambda^-_e) \tag{A.12f}$$

$$- W_P \sum_{s \in S_e^{VAL}} P_{es} x_{es} \tag{A.12g}$$

$$- W_R \sum_{r \in R} \min(V^R_r - \underline{V}^R, \overline{V}^R) w_{er} \qquad \forall e \in \mathcal{E} \tag{A.12h}$$

$$g \geq f_e \qquad \forall e \in \mathcal{E} \tag{A.13}$$

$$\sum_{s \in \mathcal{S}^{\mathcal{VAL}}} V^S_s x_{es} + \lambda^+_e - \lambda^-_e = |\mathcal{J}| B_e \quad \forall e \in \mathcal{E} \tag{A.14}$$

$$\gamma_{ei} - \gamma_{e(i+1)} = \rho^{SAT}_{ei} - \rho^{SUN}_{e(i+1)} \quad \forall e \in \mathcal{E}, i \in \mathcal{I}^{\mathcal{SAT}} \tag{A.15}$$

$$\gamma_{ei} - \gamma_{e(i-1)} - \gamma_{e(i+1)} \leq \sigma_{ei} \quad \forall e \in \mathcal{E}, i \in \{2, 3, ..., |\mathcal{I}| - 1\} \tag{A.16}$$

$$\gamma_{e(i-1)} - \gamma_{ei} + \gamma_{e(i+1)} - 1 \leq \phi_{ei} \quad \forall e \in \mathcal{E}, i \in \{2, 3, ..., |\mathcal{I}| - 1\} \tag{A.17}$$

$$\sum_{i'=i}^{i+L^{CD}} \gamma_{ei'} - L^{CD} \leq \pi_{ei} \quad \forall e \in \mathcal{E}, i \in \{1, 2, ..., |\mathcal{I}| - L^{CD}\} \tag{A.18}$$

Variable definition constraints

$$x_{es} \in \{0,1\} \qquad \forall e \in \mathcal{E}, s \in \mathcal{S}_e^{VAL} \tag{A.19}$$

$$y_{etc} \in \{0,1\} \qquad \forall e \in \mathcal{E}, t \in \mathcal{T}, c \in \mathcal{C}_e \tag{A.20}$$

$$w_{er} \in \{0,1\} \qquad \forall e \in \mathcal{E}, r \in \mathcal{R} \tag{A.21}$$

$$\delta_{tc}^{+}, \delta_{tc}^{-} \in \mathbb{Z}^{+} \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{A.22}$$

$$\rho_{ei}^{SAT}, \rho_{e(i+1)}^{SUN} \in \{0,1\} \qquad \forall e \in \mathcal{E}, i \in \mathcal{I}^{\mathcal{SAT}} \tag{A.23}$$

$$\gamma_{ei}, \sigma_{ei}, \phi_{ei}, \pi_{ei} \in \{0,1\} \qquad \forall e \in \mathcal{E}, i \in \mathcal{I} \tag{A.24}$$

$$\mu_{tc} \in \mathbb{Z}^{+} \qquad \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{A.25}$$

$$\lambda_e^{+}, \lambda_e^{-} \ \geq 0 \qquad \forall e \in \mathcal{E} \tag{A.26}$$

$$f_e \ \ free \qquad \forall e \in \mathcal{E} \tag{A.27}$$

$$g \ \ free \tag{A.28}$$

Appendix B

Operators and Parameter Values

In this appendix we present the operators and parameter values that we use in the computational study. Section B.1 contains all of the operators and possible combinations, and Section B.2 presents parameter values.

B.1 Operators

Table B.1: Naming convention for the destroy operator types.

Destroy Operator Type	**Symbol**
Uniform Random Week	RW
Greedy Week	GW
Weighted Random Week	WW
Uniform Random Employee	RE
Greedy Employee	GE
Weighted Random Employee	WE
Weekend	WKD
Partial Weekend	PW
Demand Day	DD
Demand Shift	DS
No Destroy	ND

Table B.2: Naming convention for selection strategies of the repair operators.

Repair Selection Strategy	**Symbol**
Greedy Employee Selection	GES
Weighted Employee Selection	WES
Random Employee Selection	RES
Greedy Shift Selection	GSS
Weighted Shift Selection	WSS
Random Shift Selection	RSS

Table B.3: Destroy/Repair Pairs used in the computational study.

Operator Number	Destroy Operator	Repair Operator	k
0	RW	GES/GSS	1
1	RW	WES/GSS	1
2	GW	GES/GSS	1
3	GW	WES/GSS	1
4	WW	GES/GSS	1
5	WW	WES/GSS	1
6	RE	WES/GSS	2
7	RE	GES/GSS	2
8	GE	WES/GSS	2
9	GE	GES/GSS	2
10	WE	WES/GSS	2
11	WE	GES/GSS	2
12	WKD	WES/GSS	1
13	PW	WES/WSS	-
14	DD	WES/RSS	-
15	DD	GES/WSS	-
16	DD	WES/WSS	-
17	DS	GES/GSS	0.5
26	ND	WES/GSS	-

Table B.4: Hybrid operators used in the computational study.

Number	Type	Employee Selection	Day/Shift Selection	Size
18	Swap	Random	Random	1
19	Swap	Random	Random	5
20	Swap	Greedy	Random	1
21	Swap	Greedy	Greedy	1
22	Swap	Targeted	Targeted	-
23	Change	Targeted	Targeted	-
24	Change	Random	Random	5
25	Change	Greedy	Random	1

B.2 Parameters

B.2.1 Problem Parameters

Table B.5: Table of objective function weights.

Weight		**Value**
Lowest individual fairness	W_F	0.1
Weekly rest duration	W_R	0.5
Contracted hours difference	W_B	1
Partial weekends	W_{PW}	8
Isolated work day	W_{IW}	10
Isolated off day	W_{IO}	10
Consecutive days	W_{CD}	12
Preference	W_P	0.1
Shift duration	W_{SL}	1
Demand coverage	W_D	0.01

B.2.2 Training Parameters

Table B.6: Table of training parameters.

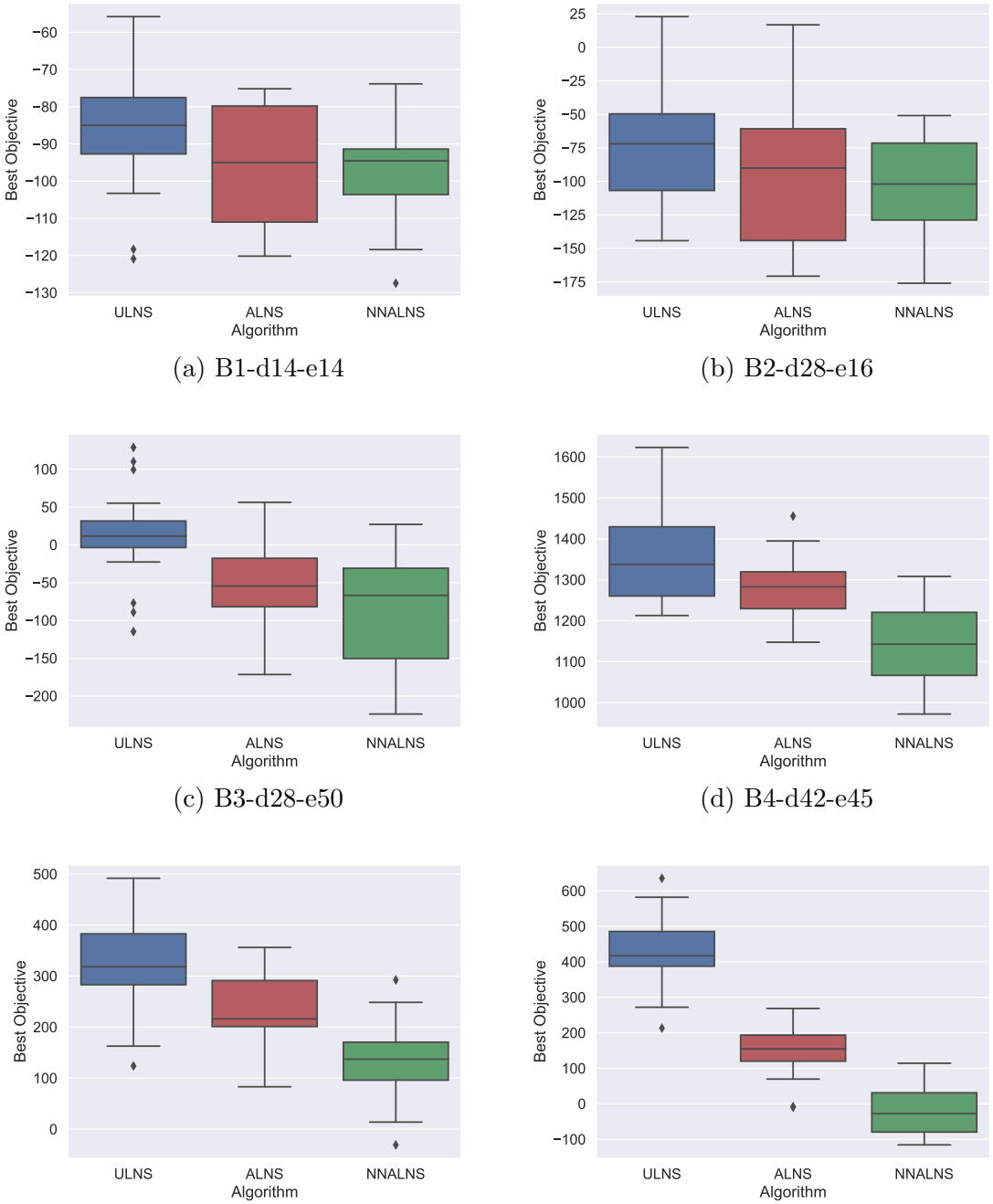
Training parameters	
Total # of episodes	1000
Episodes before update (E)	20
Number of updates (U)	50
Learning rate actor	0.0003
Learning rate critic	0.001
Optimizer	Adam
Epochs (N)	50
Epslion for L^{CLIP} loss (ϵ)	0.2
Discount factor (γ)	0.99
Exponential weight discount (λ)	0.97
Mini-batch size (% of total)	0.01
Critic loss weight (β)	0.5
Entropy bonus weight (μ)	0.01

Appendix C

Computational Study

We only present a subset of the figures in the computational study in Chapter 8. In this appendix we present additional figures and information relevant to the computational study.

C.1 Comparative Study



(a) B1-d14-e14

(b) B2-d28-e16

(c) B3-d28-e50

(d) B4-d42-e45

(e) B5-d56-e20

(f) B6-d84-e22

Figure C.1: Box plot of best objective value for benchmark instances B1 through B6 after 1000 iterations of NNALNS, ALNS and ULNS over 20 runs.

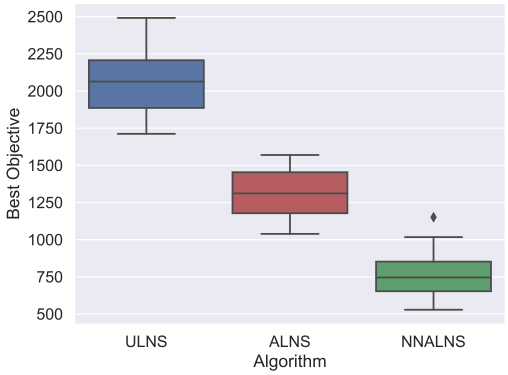
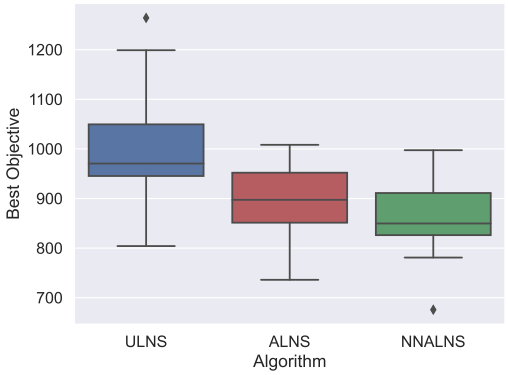
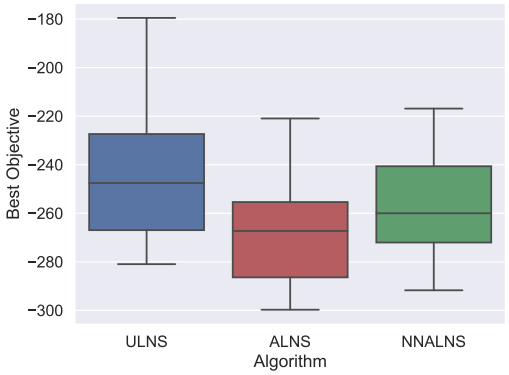


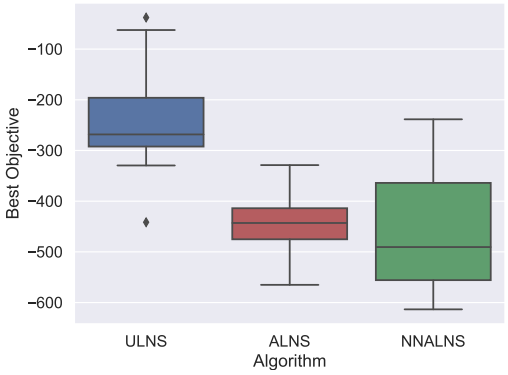
Figure C.2: Box plot of best objective value for B7-d182-e50 after 1000 iterations of NNALNS, ALNS and ULNS over 20 runs.



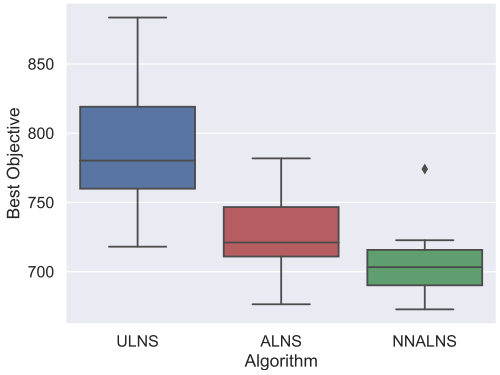
(a) V1-d56-e9



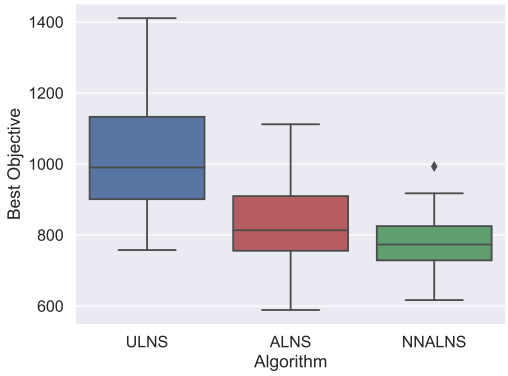
(b) V2-d70-e6



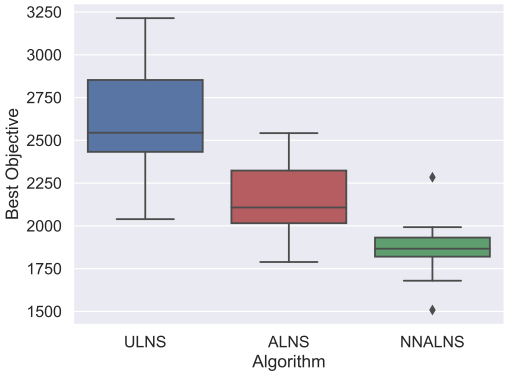
(c) V3-d46-e28



(d) V4-d84-e8

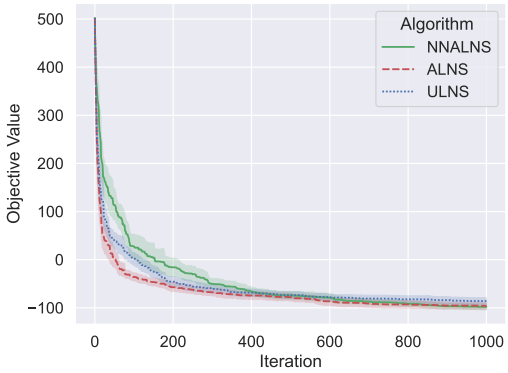


(e) V5-d42-e15

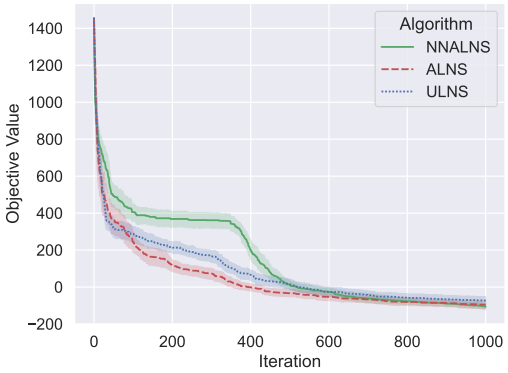


(f) V6-d98-e15

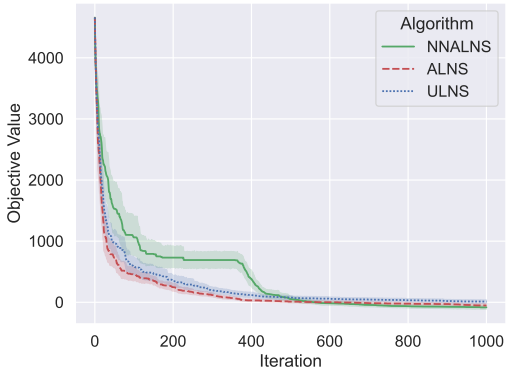
Figure C.3: Box plot of best objective value for Visma instances after 1000 iterations of NNALNS, ALNS and ULNS over 20 runs.



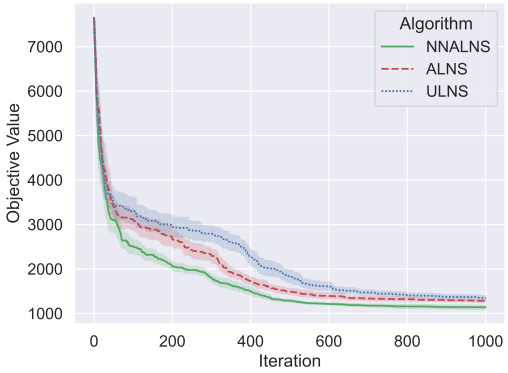
(a) B1-d14-e14



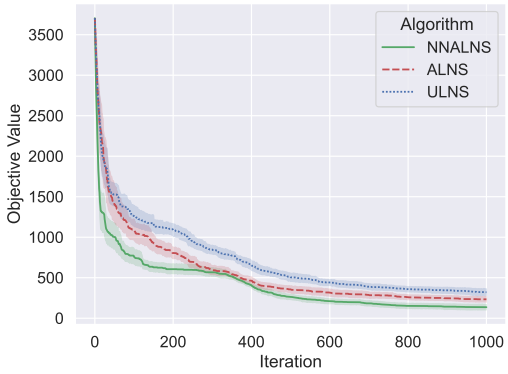
(b) B2-d28-e16



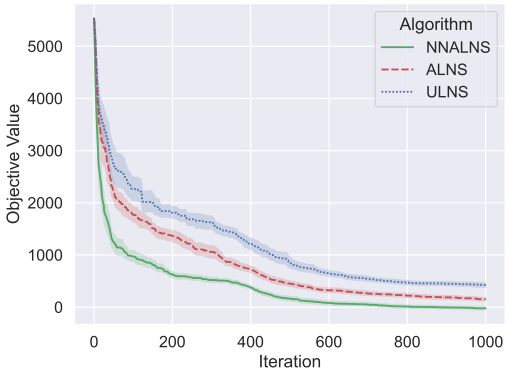
(c) B3-d28-e50



(d) B4-d42-e45



(e) B5-d56-e20



(f) B6-d84-e22

Figure C.4: Average best objective value over 20 runs with a 95% confidence interval for benchmark instances B1 through B6 using NNALNS, ALNS and ULNS.

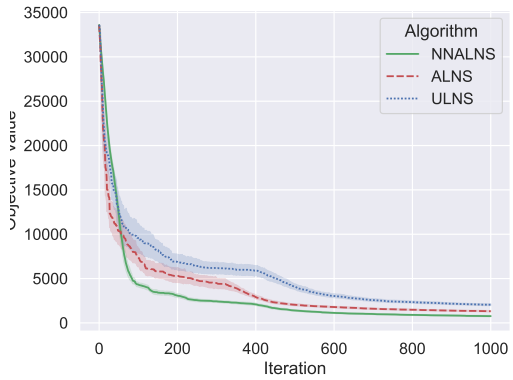
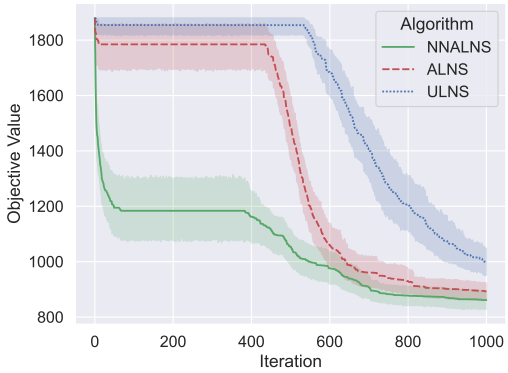
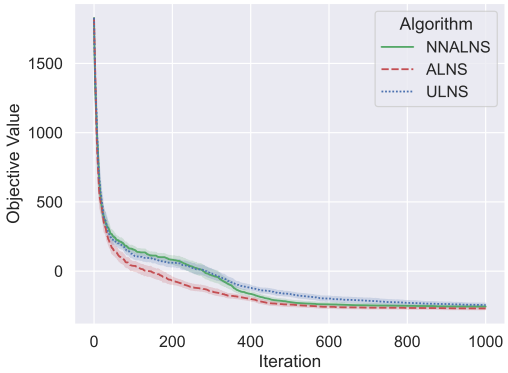


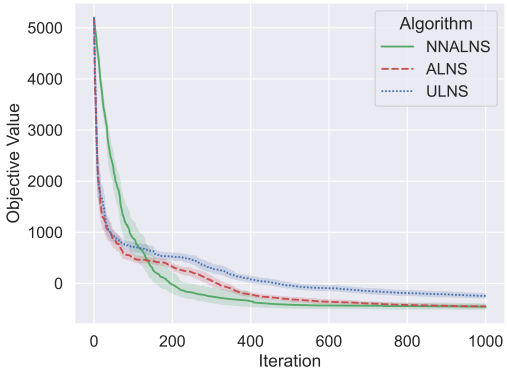
Figure C.5: Average best objective value over 20 runs with a 95% confidence interval for B7-d182-e50 using NNALNS, ALNS and ULNS.



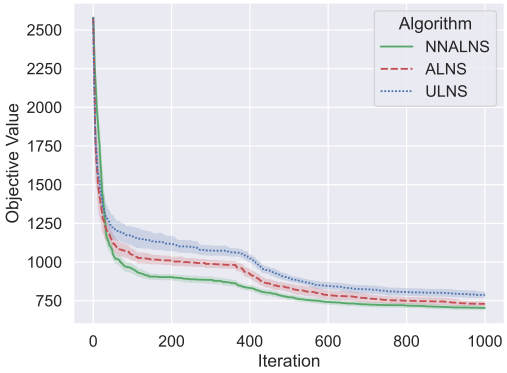
(a) V1-d56-e9



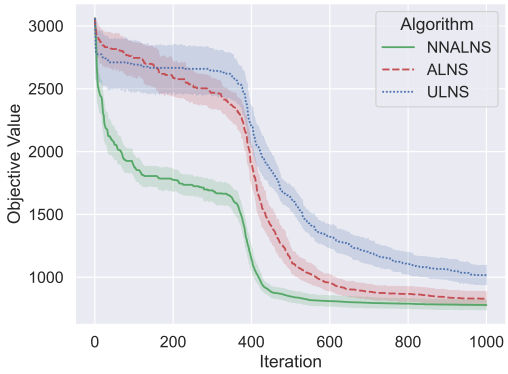
(b) V2-d70-e6



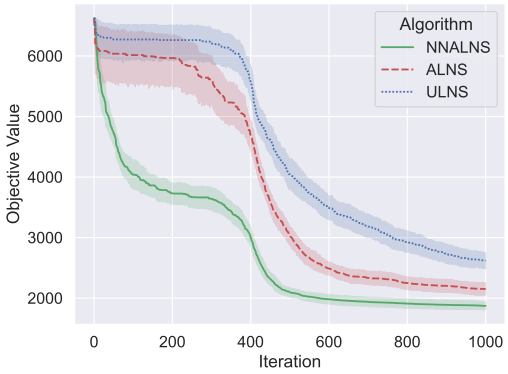
(c) V3-d46-e28



(d) V4-d84-e8



(e) V5-d42-e15

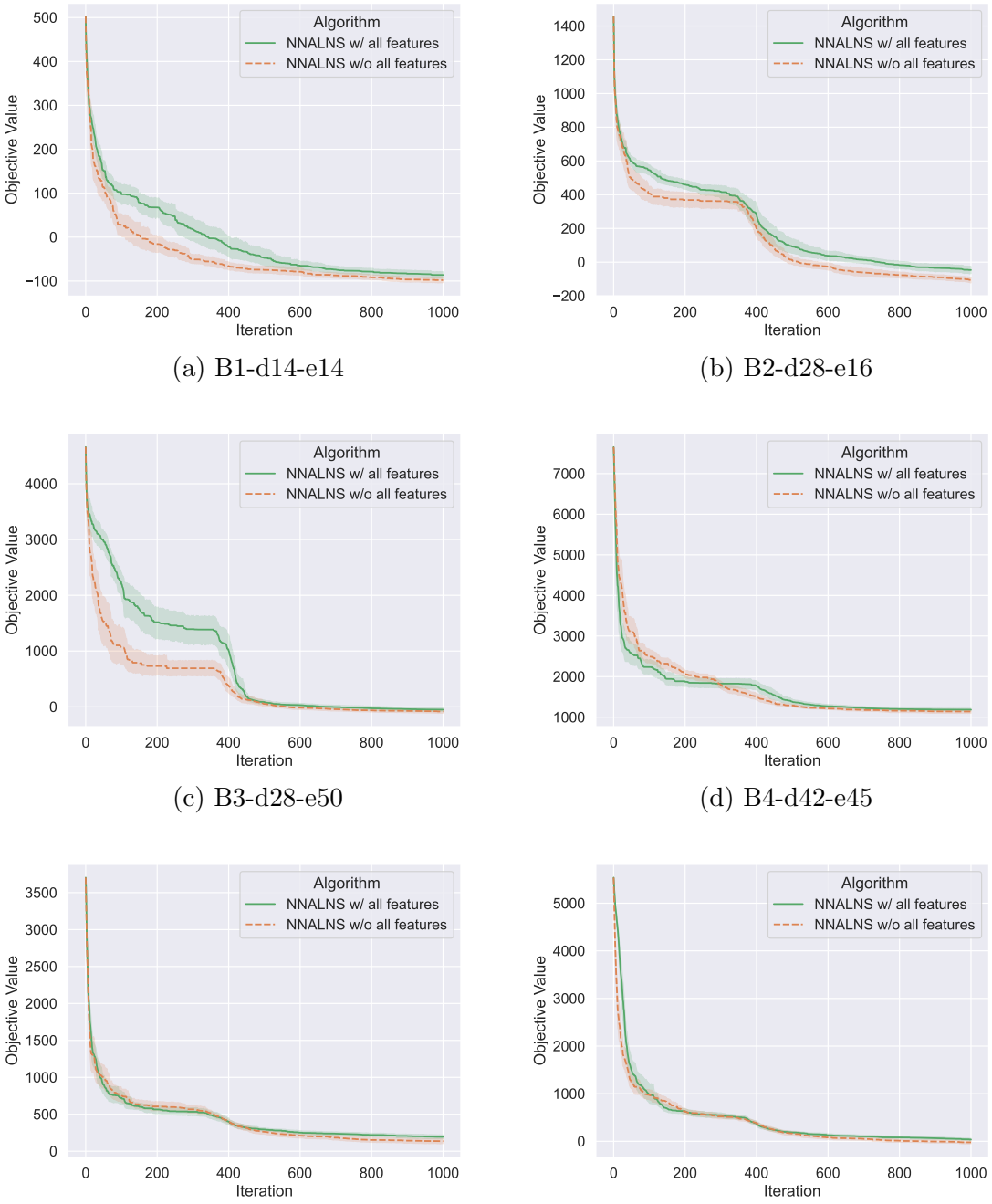


(f) V6-d98-e15

Figure C.6: Average best objective value over 20 runs with a 95% confidence interval for Visma instances using NNALNS, ALNS and ULNS.

C.2 Value of Problem-Specific Features

C.2.1 Objective Value



(a) B1-d14-e14

(b) B2-d28-e16

(c) B3-d28-e50

(d) B4-d42-e45

(e) B5-d56-e20

(f) B6-d84-e22

Figure C.7: Average best objective value over 20 runs with a 95% confidence interval for the benchmark instances B1 through B6 using NNALNS with and without problem-specific features.

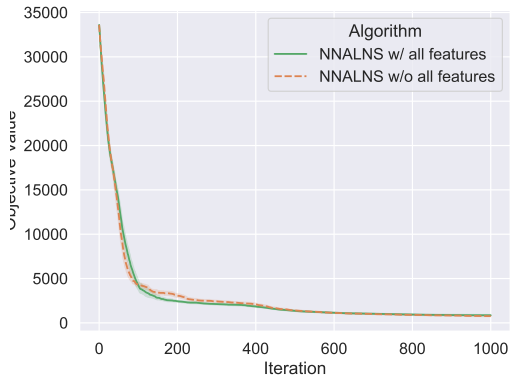
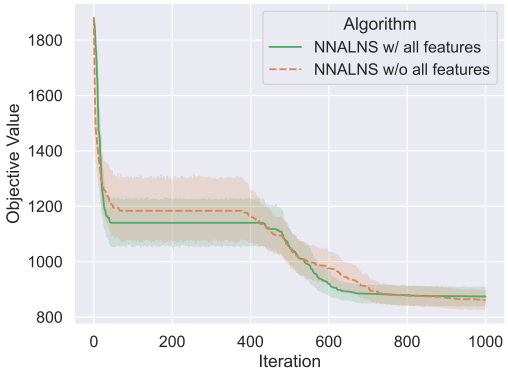
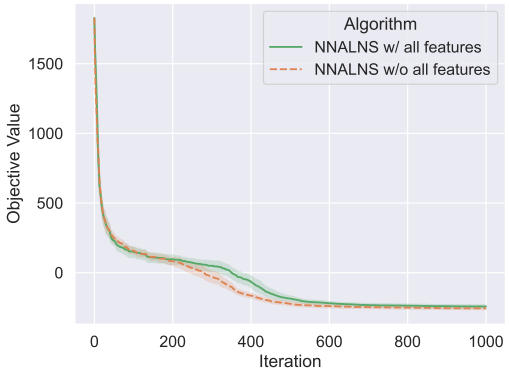


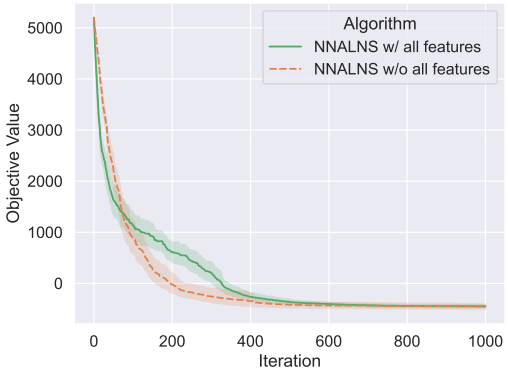
Figure C.8: Average best objective value over 20 runs with a 95% confidence interval for benchmark instance B7-d182-e50 using NNALNS with and without problem-specific features.



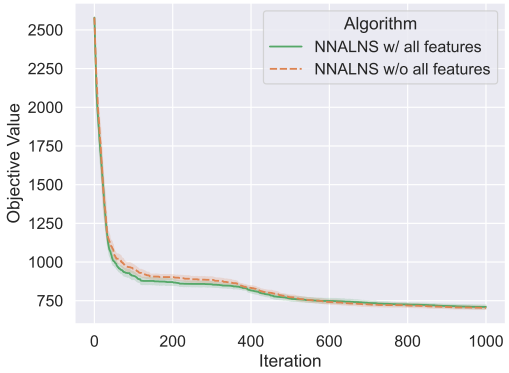
(a) V1-d56-e9



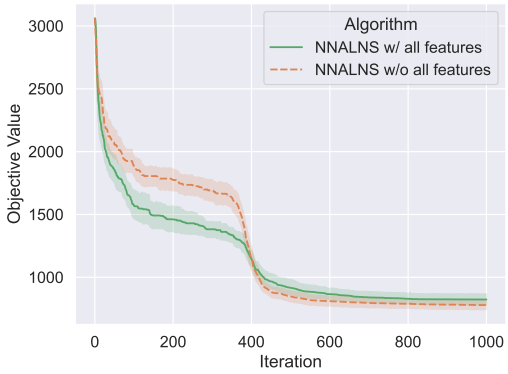
(b) V2-d70-e6



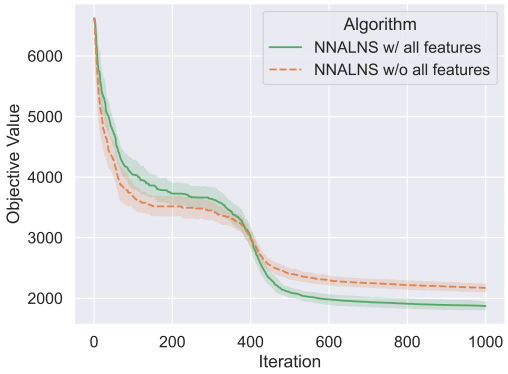
(c) V3-d46-e28



(d) V4-d84-e8



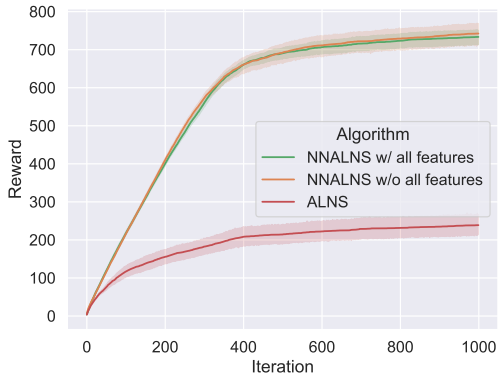
(e) V5-d42-e15



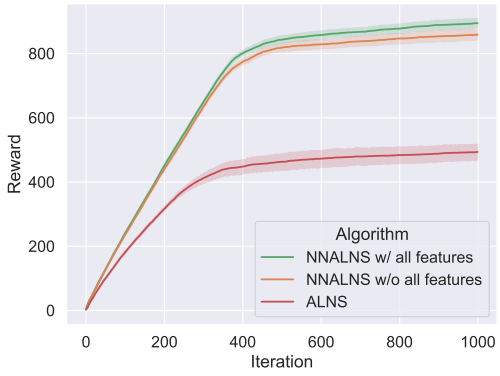
(f) V6-d98-e15

Figure C.9: Average best objective value over 20 runs with a 95% confidence interval for Visma instances V1 through V6 using NNALNS with and without problem-specific features

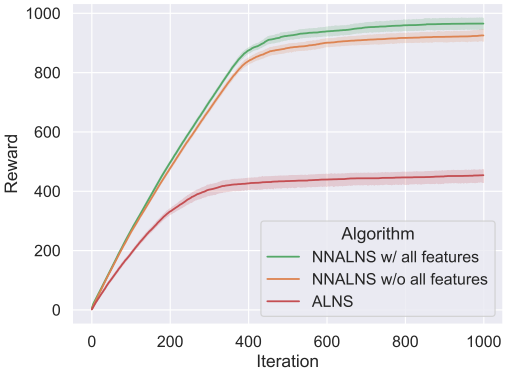
C.2.2 Reward



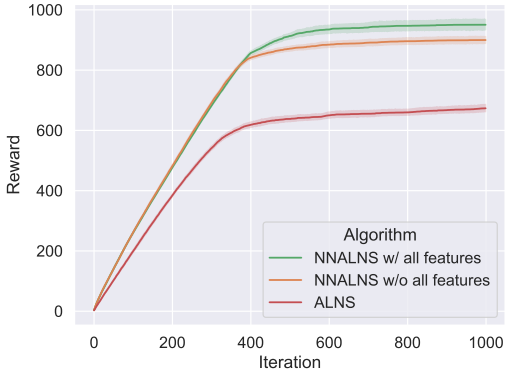
(a) B1-d14-e14



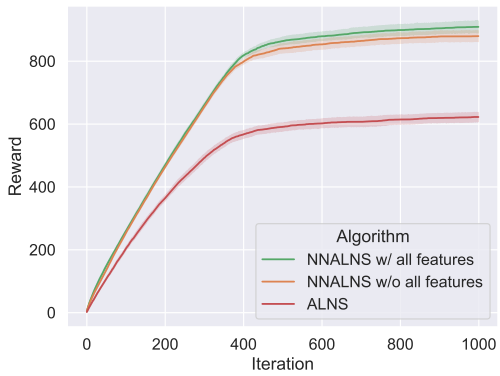
(b) B2-d28-e16



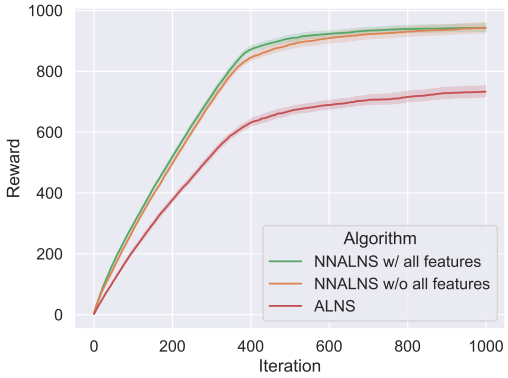
(c) B3-d28-e50



(d) B4-d42-e45



(e) B5-d56-e20



(f) B6-d84-e22

Figure C.10: Average cumulative reward over 20 runs with a 95% confidence interval for benchmark instances B1 through B6 using NNALNS with and without problem-specific features and ALNS.

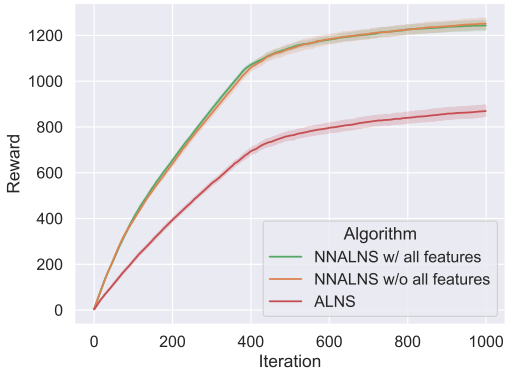
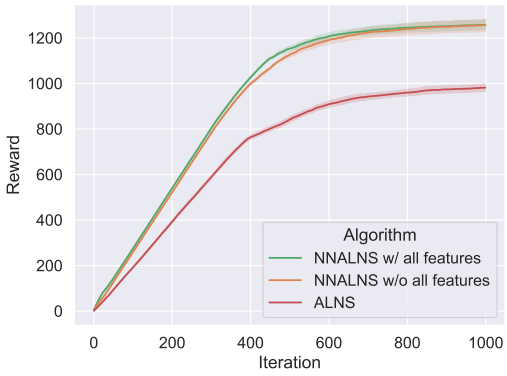
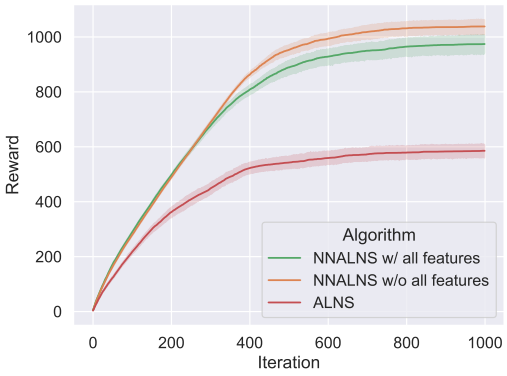


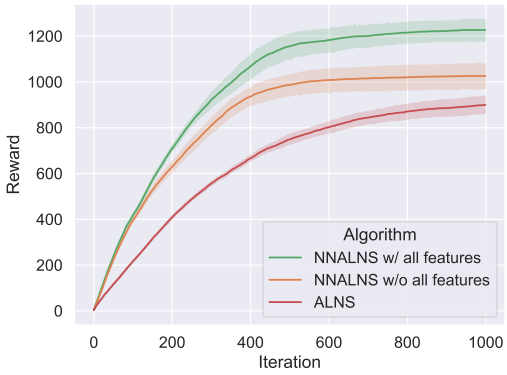
Figure C.11: Average cumulative reward over 20 runs with a 95% confidence interval for benchmark instances B7-d182-e50 using NNALNS with and without problem-specific features and ALNS.



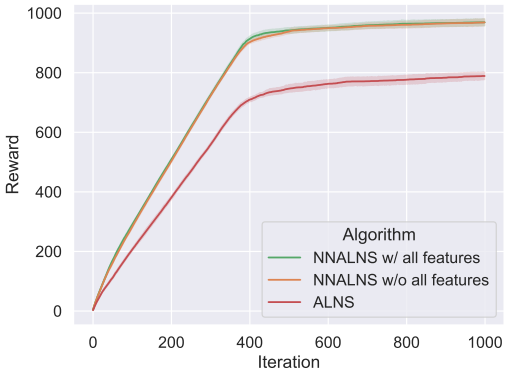
(a) V1-d56-e9



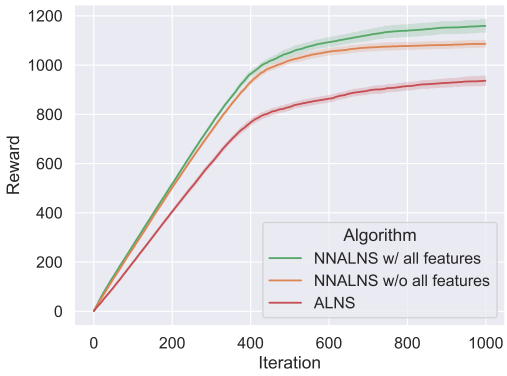
(b) V2-d70-e6



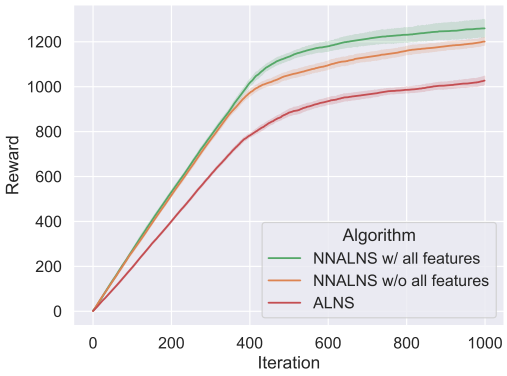
(c) V3-d46-e28



(d) V4-d84-e8



(e) V5-d42-e15

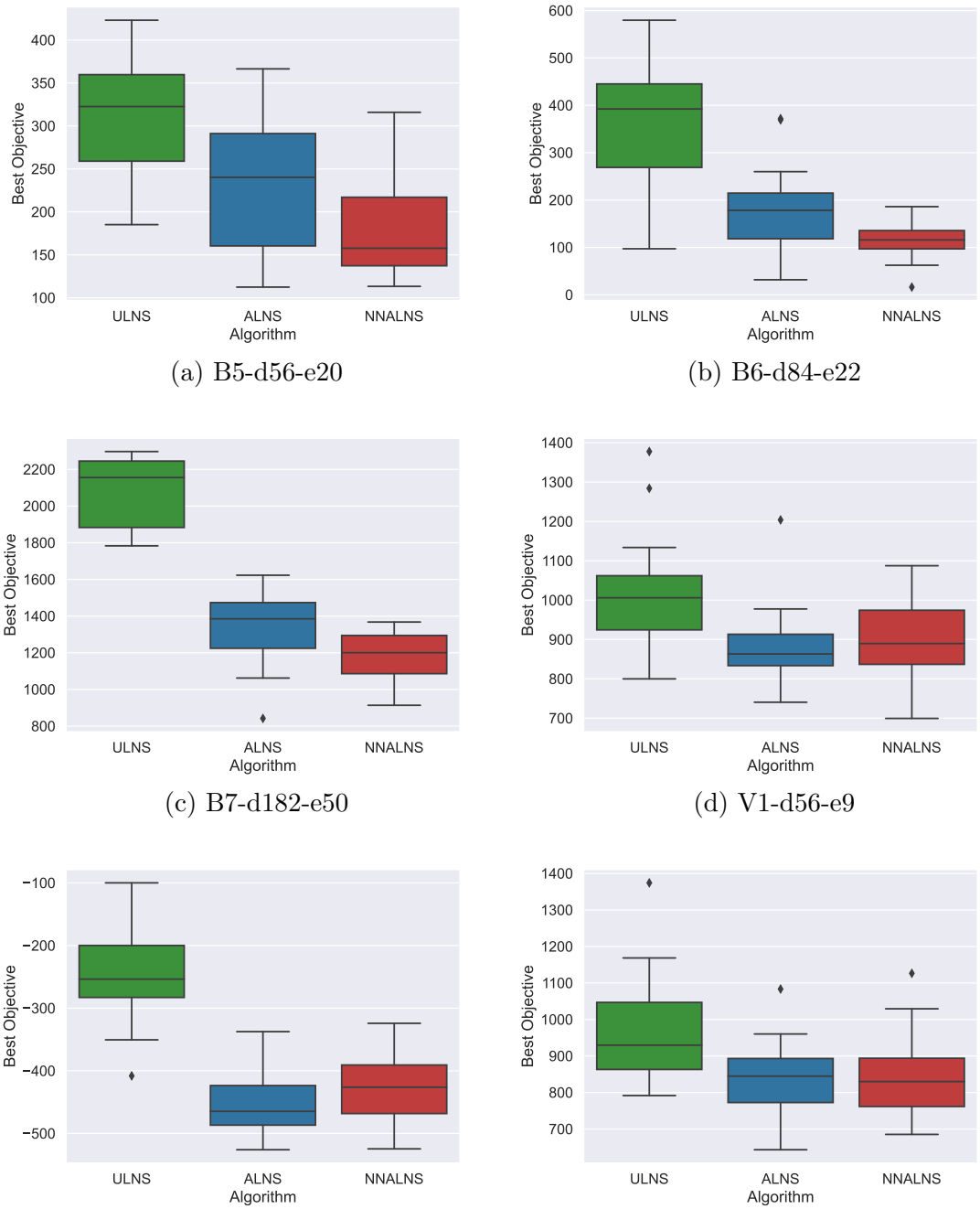


(f) V6-d98-e15

Figure C.12: Average cumulative reward over 20 runs with a 95% confidence interval for Visma instances V1 through V6 using NNALNS with and without problem-specific features and ALNS.

C.3 Generalized Learning

C.3.1 Cross-Instance Learning



(a) B5-d56-e20

(b) B6-d84-e22

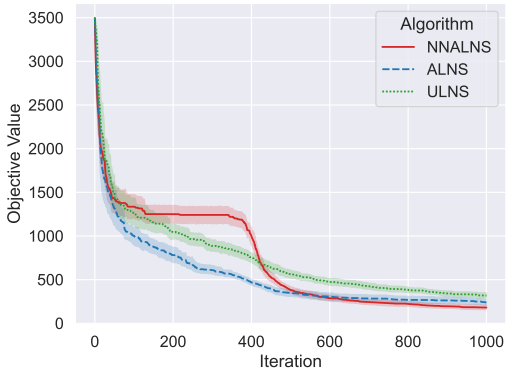
(c) B7-d182-e50

(d) V1-d56-e9

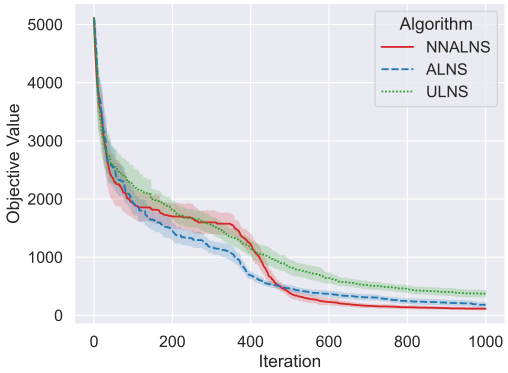
(e) V3-d46-e28

(f) V5-d42-e15

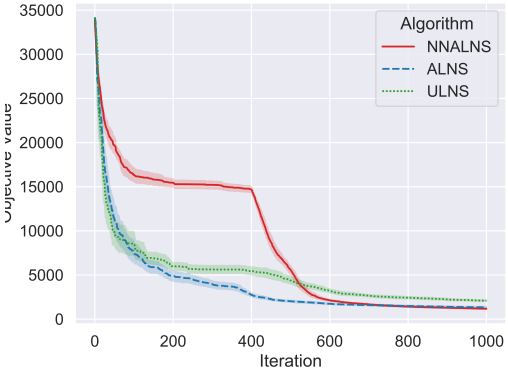
Figure C.13: Box plot of best objective value over 20 runs for the test instances in the cross-instance experiments using NNALNS, ALNS and ULNS.



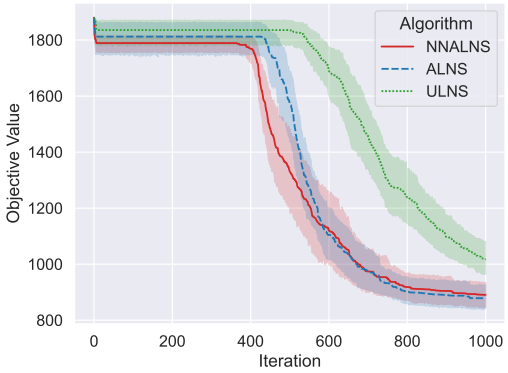
(a) B5-d56-e20



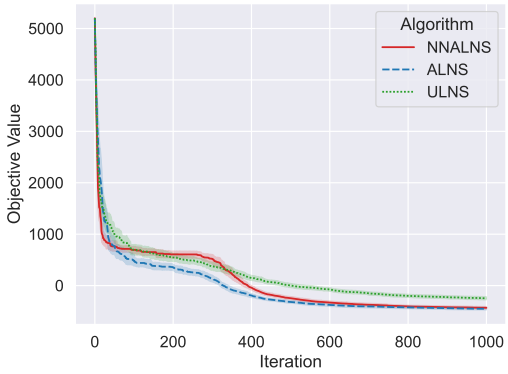
(b) B6-d84-e22



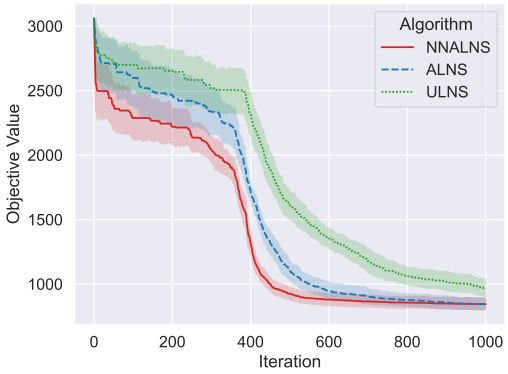
(c) B7-d182-e50



(d) V1-d56-e9



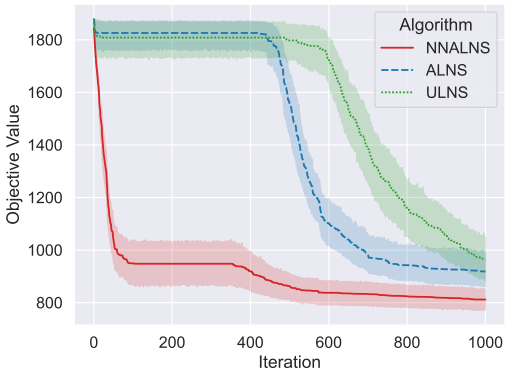
(e) V3-d46-e28



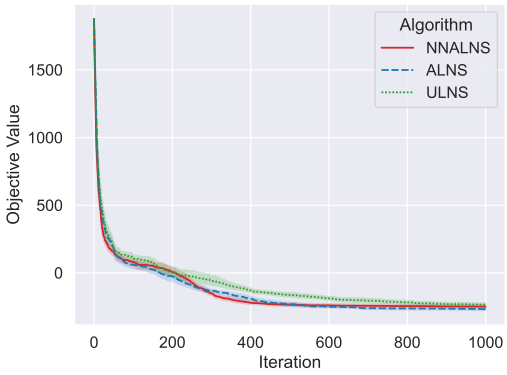
(f) V5-d42-e15

Figure C.14: Average best objective value over 20 runs with a 95% confidence interval for the test instances in the cross-instance experiments using NNALNS, ALNS and ULNS.

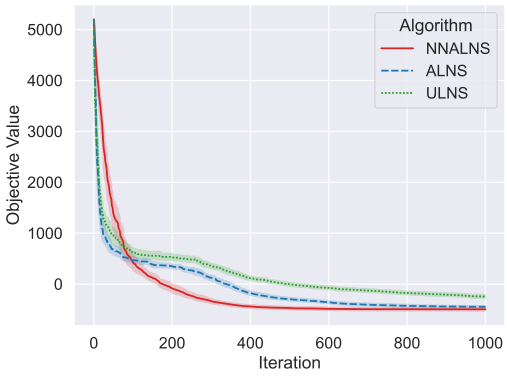
C.3.2 Adaptability



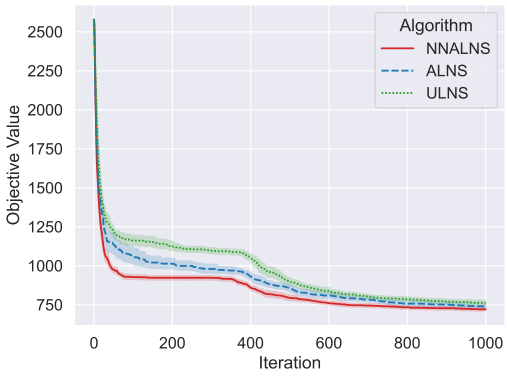
(a) V1-d56-e9



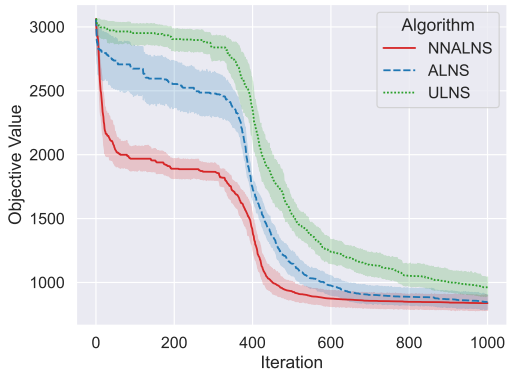
(b) V2-d70-e6



(c) V3-d46-e28

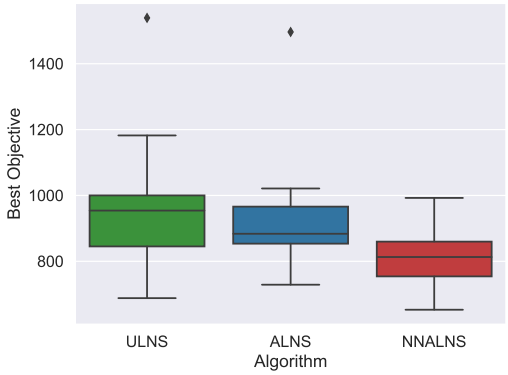


(d) V4-d84-e8

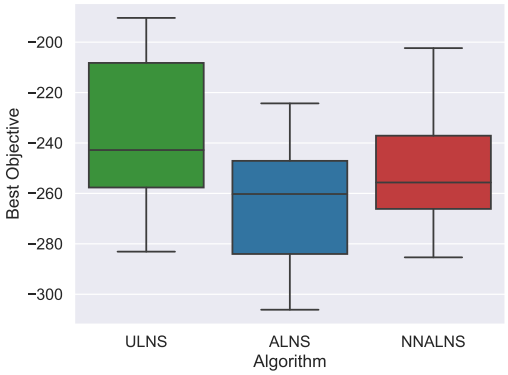


(e) V5-d42-e15

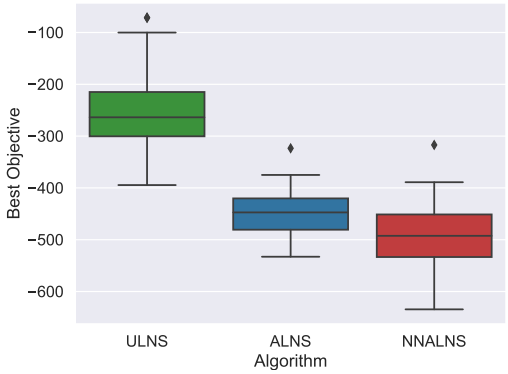
Figure C.15: Average best objective value over 20 runs with a 95% confidence interval using NNALNS, ALNS and ULNS on V1 through V5 when the policies are trained on small permutations of the instances.



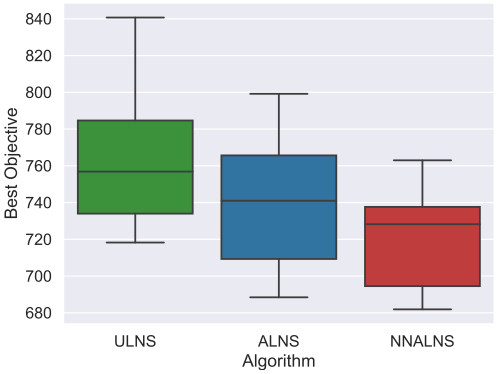
(a) V1-d56-e9



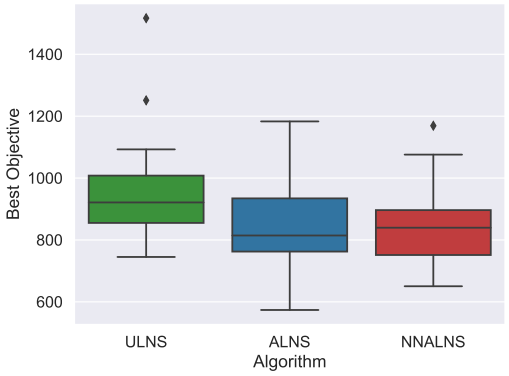
(b) V2-d70-e6



(c) V3-d46-e28



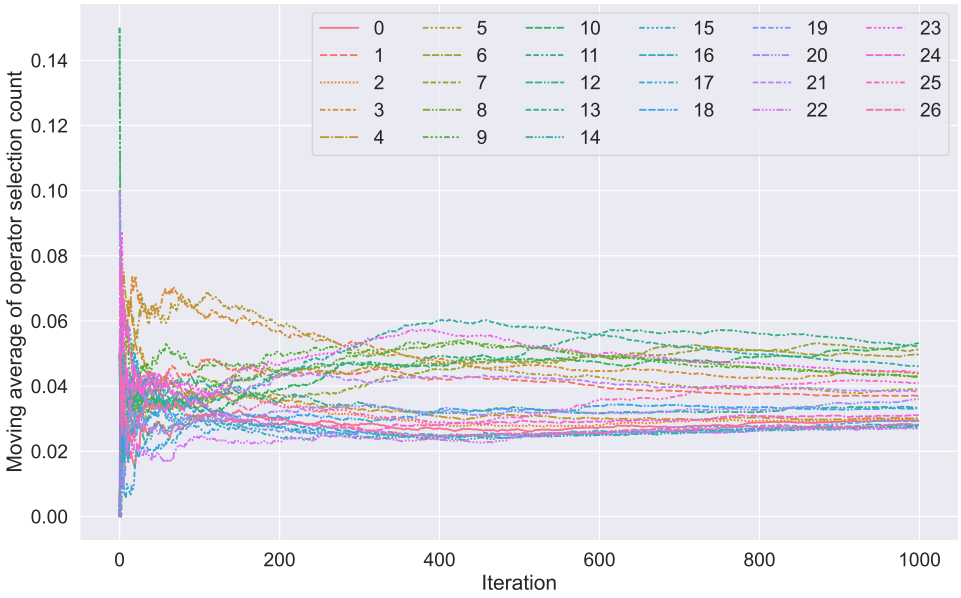
(d) V4-d84-e8



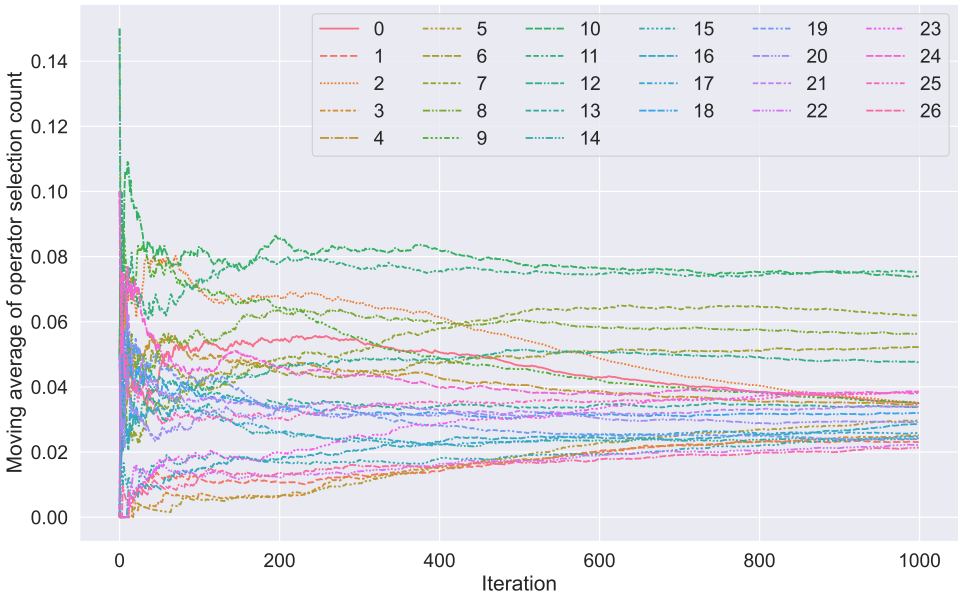
(e) V5-d42-e15

Figure C.16: Box plot of best objective value over 20 runs using NNALNS, ALNS and ULNS on V1 through V5 when the policies are trained on small permutations of the instances.

C.4 Selection Strategies

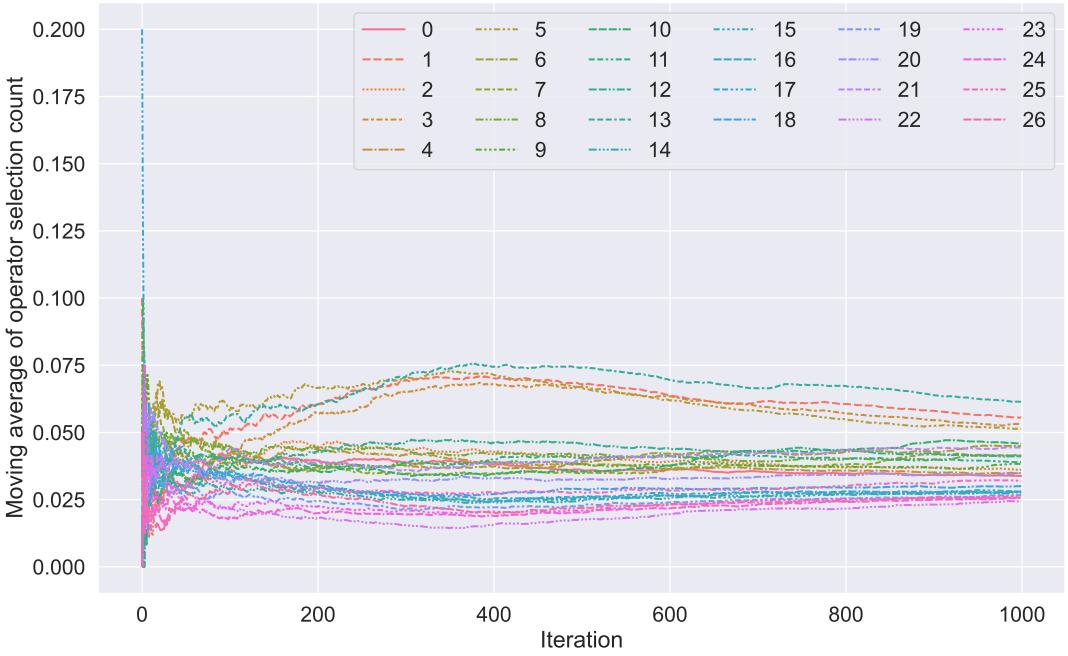


(a) ALNS

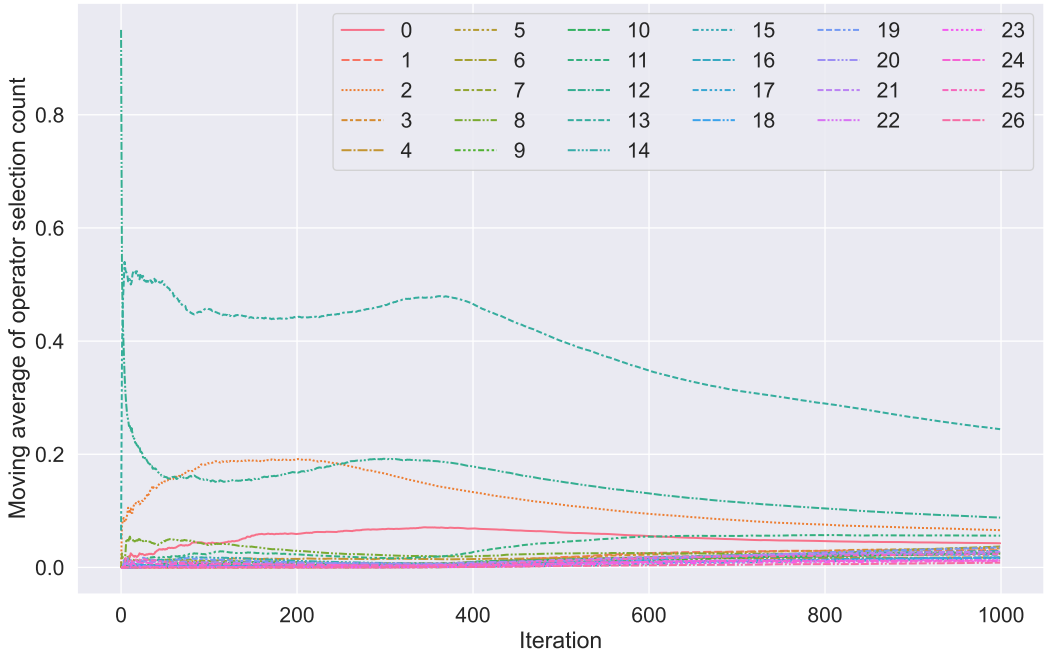


(b) NNALNS

Figure C.17: Moving average of operator selection count for ALNS and NNALNS for B1-d14-e14.

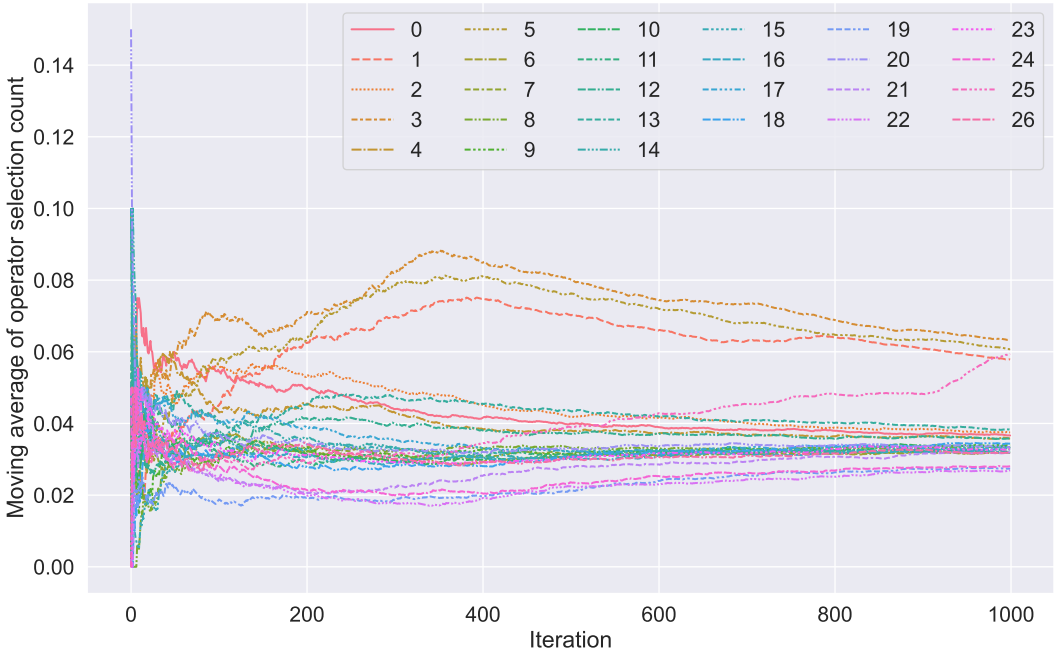


(a) ALNS

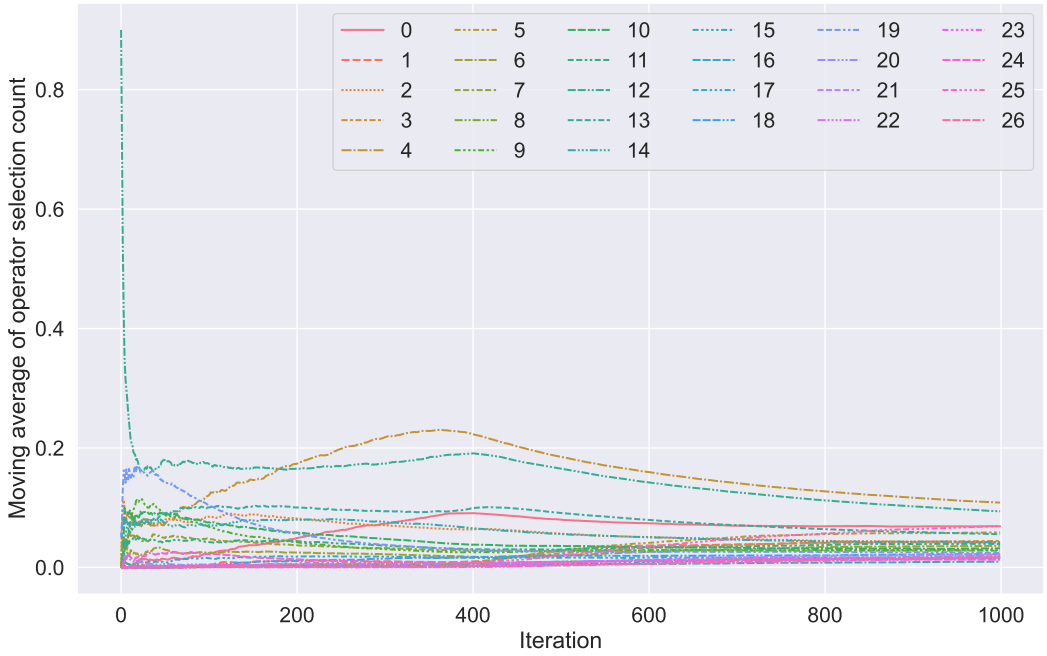


(b) NNALNS

Figure C.18: Moving average of operator selection count for ALNS and NNALNS for B2-d28-e16.

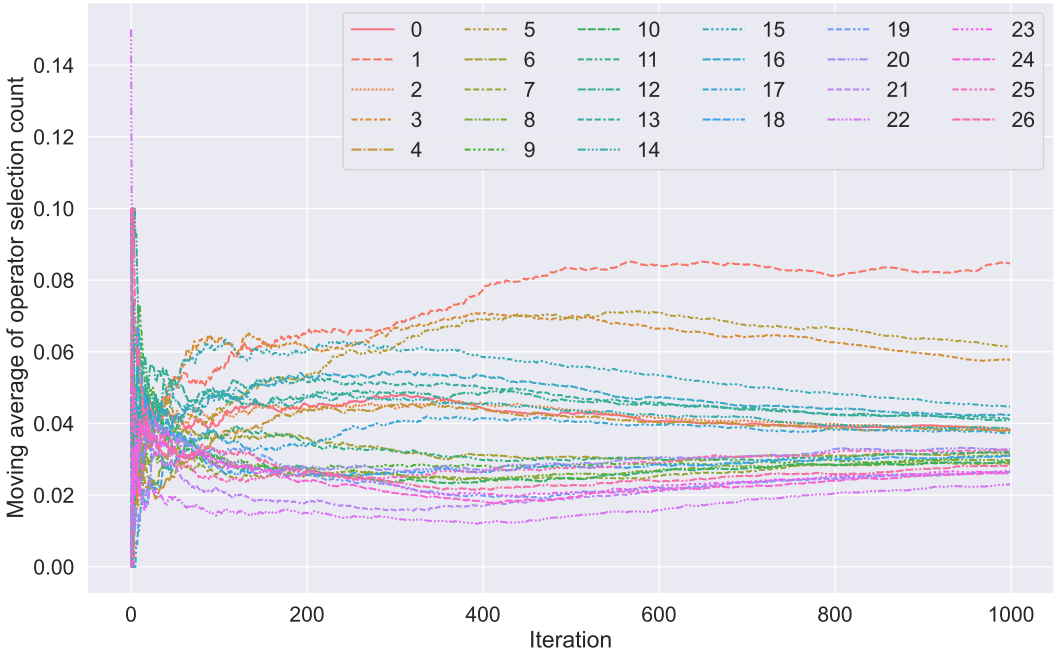


(a) ALNS

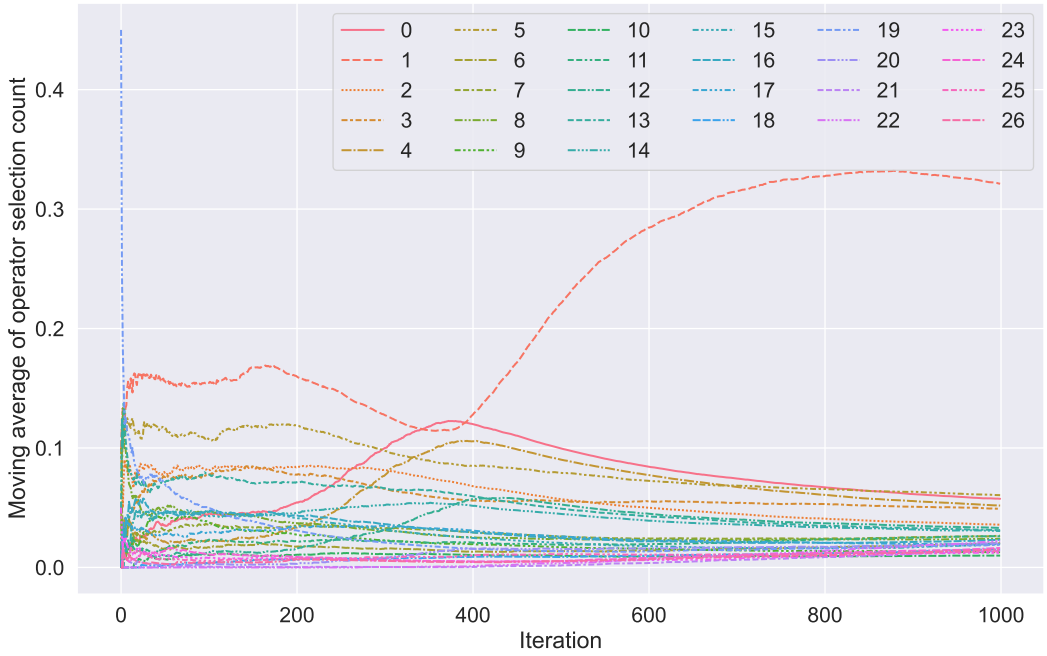


(b) NNALNS

Figure C.19: Moving average of operator selection count for ALNS and NNALNS for B3-d28-e50.

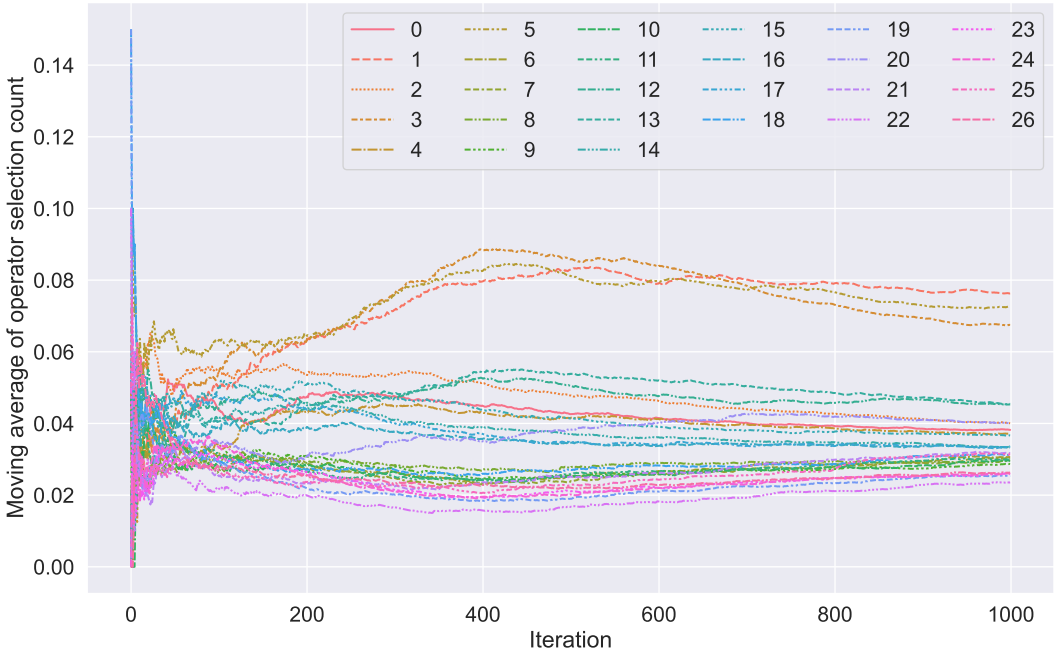


(a) ALNS

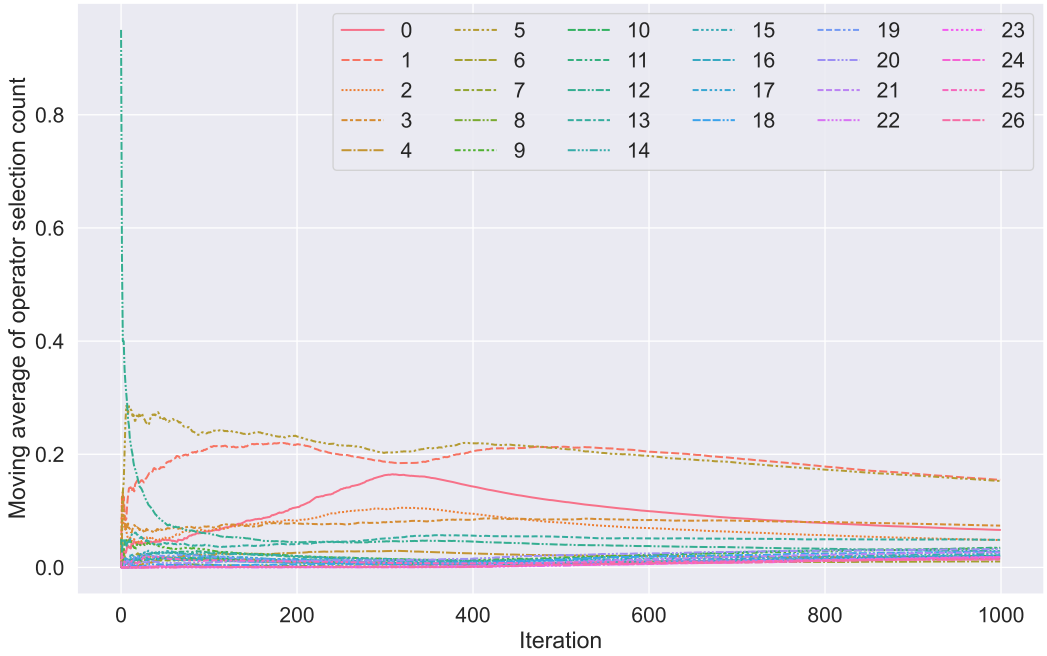


(b) NNALNS

Figure C.20: Moving average of operator selection count for ALNS and NNALNS for B4-d42-e45.

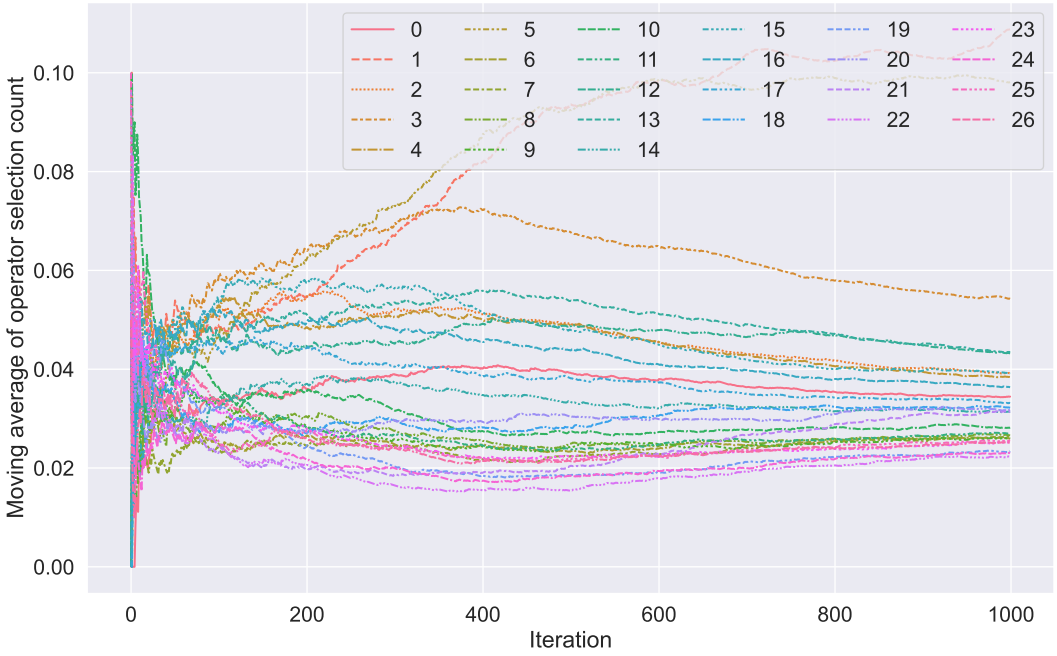


(a) ALNS

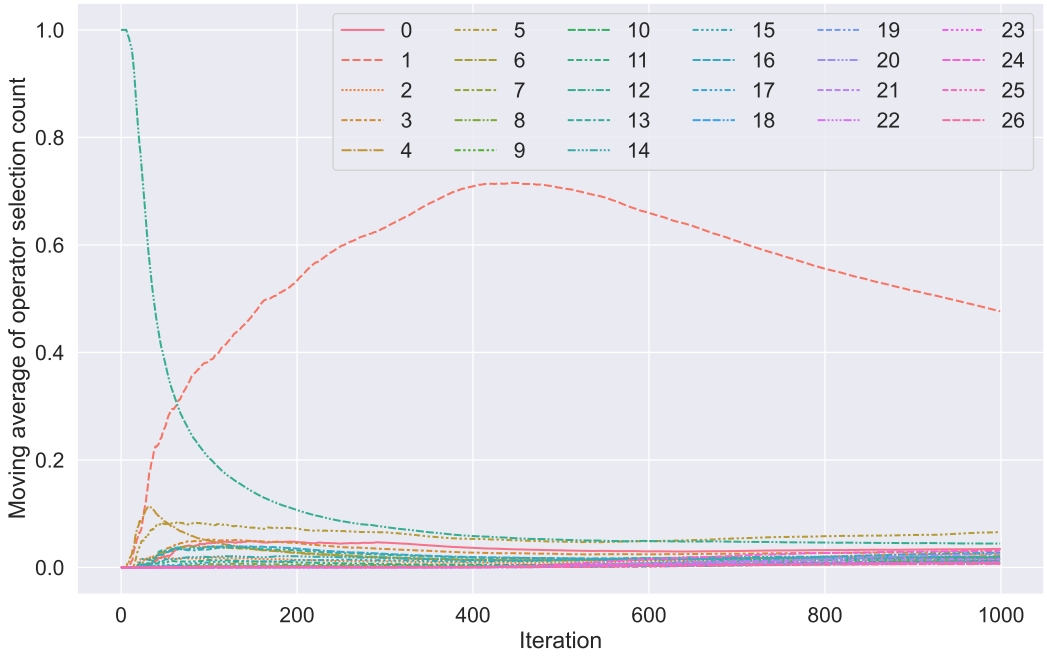


(b) NNALNS

Figure C.21: Moving average of operator selection count for ALNS and NNALNS for B5-d56-e20.

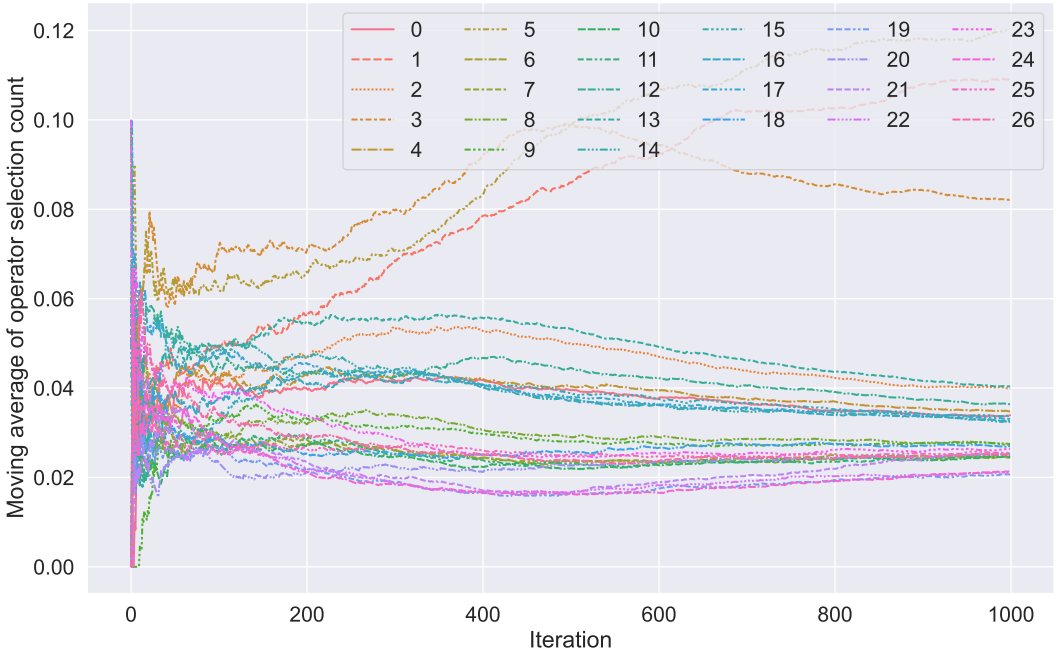


(a) ALNS

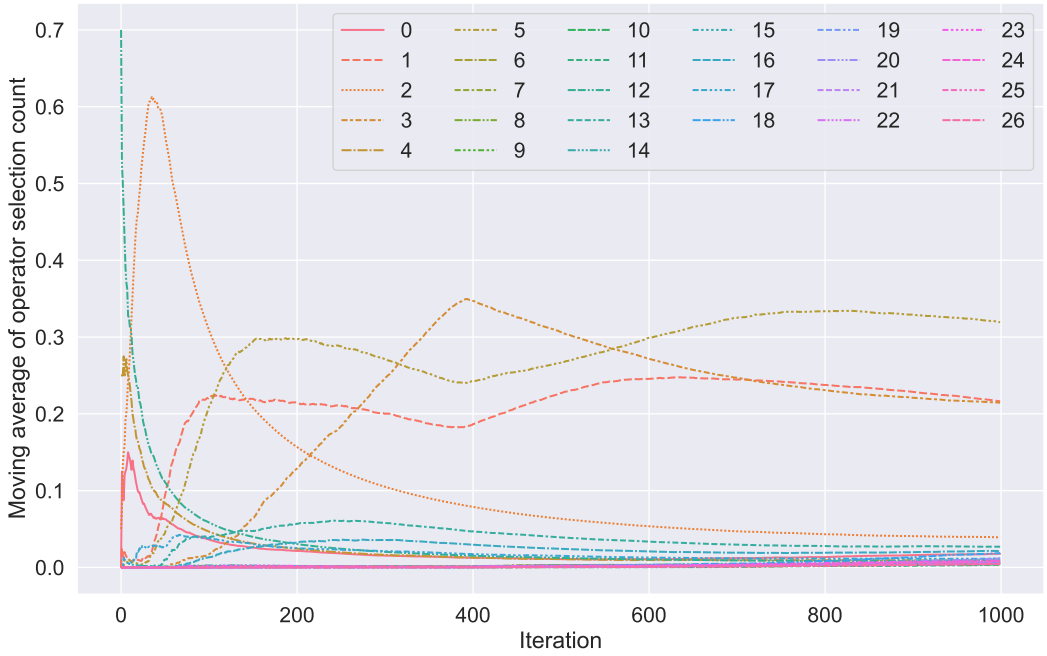


(b) NNALNS

Figure C.22: Moving average of operator selection count for ALNS and NNALNS for B6-d84-e22.

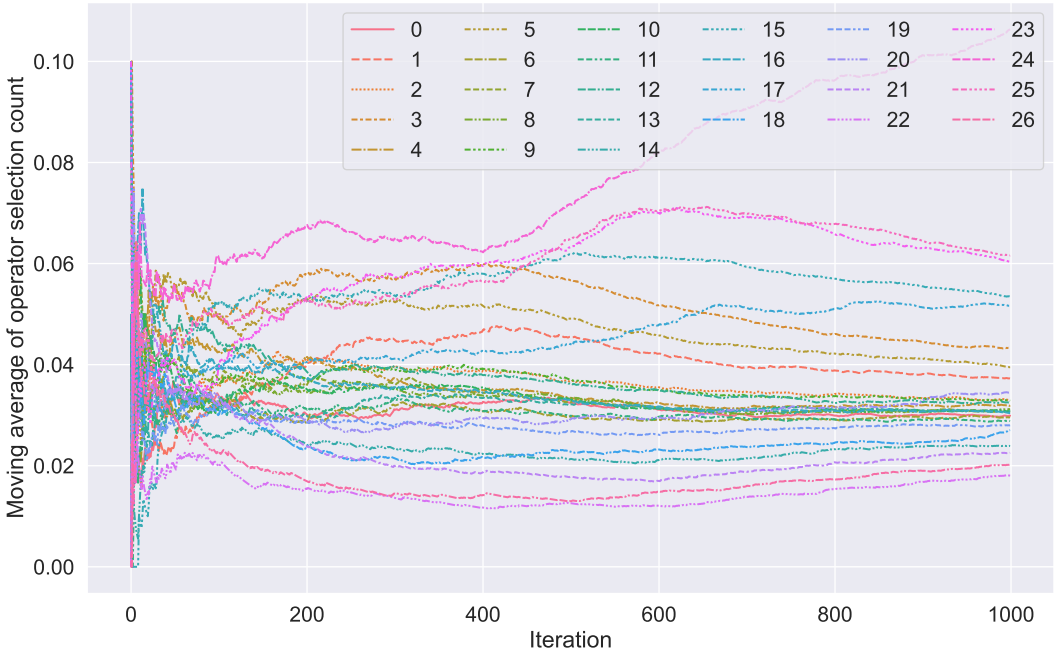


(a) ALNS

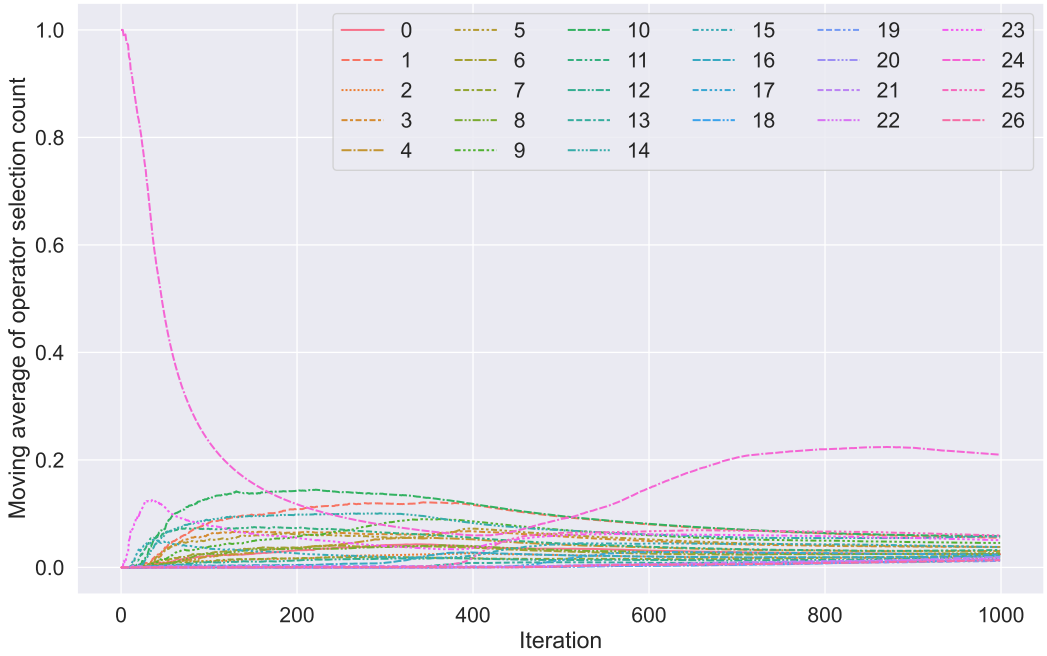


(b) NNALNS

Figure C.23: Moving average of operator selection count for ALNS and NNALNS for B7-d182-e50.

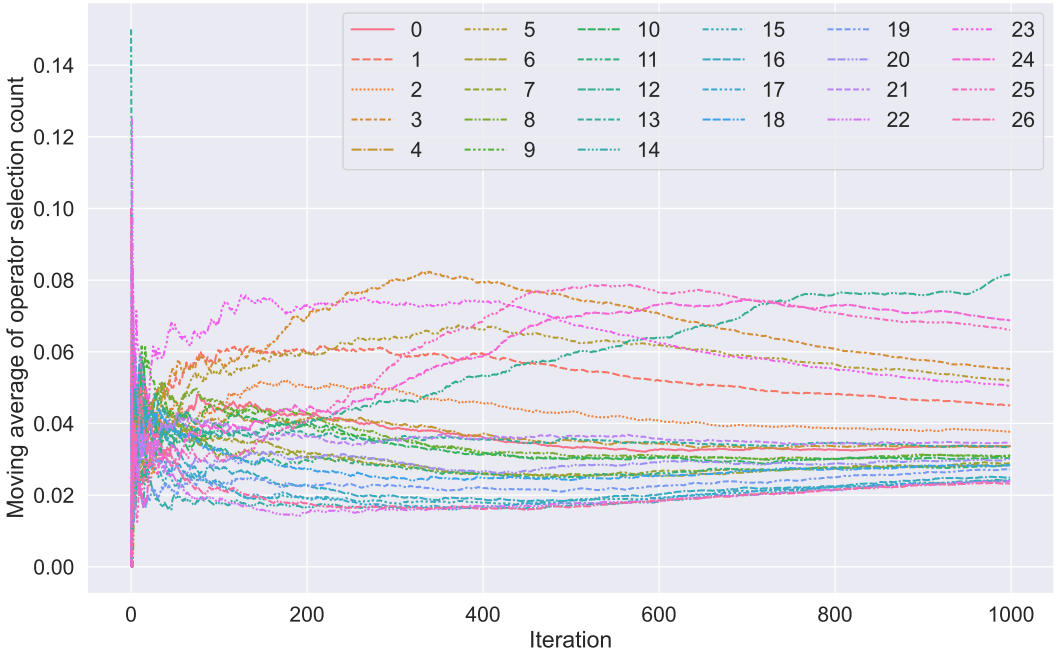


(a) ALNS

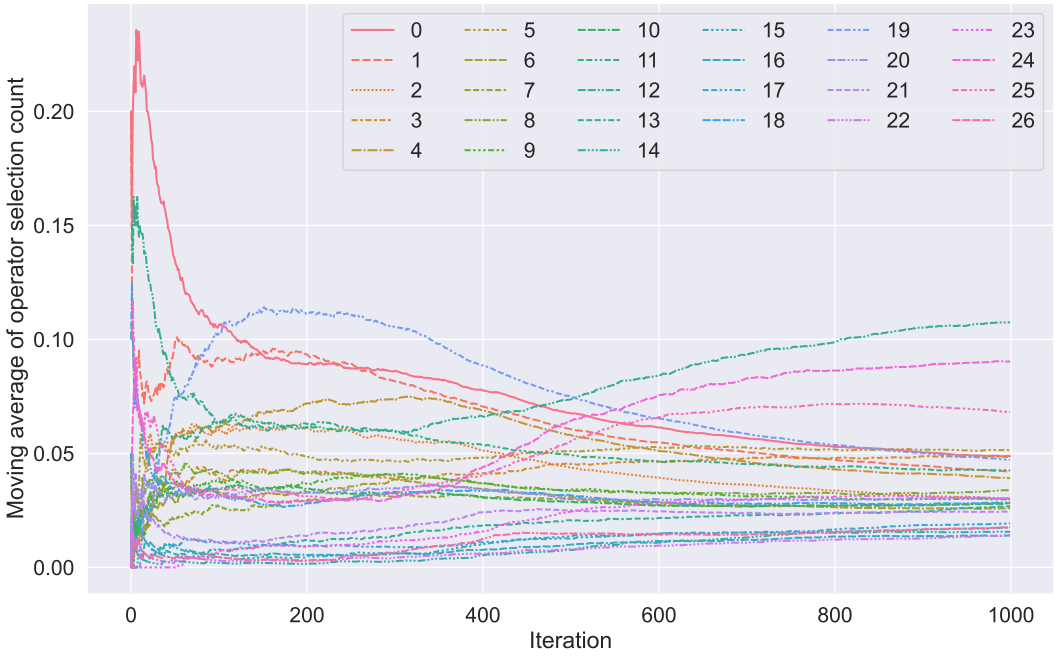


(b) NNALNS

Figure C.24: Moving average of operator selection count for ALNS and NNALNS for V1-d56-e9.

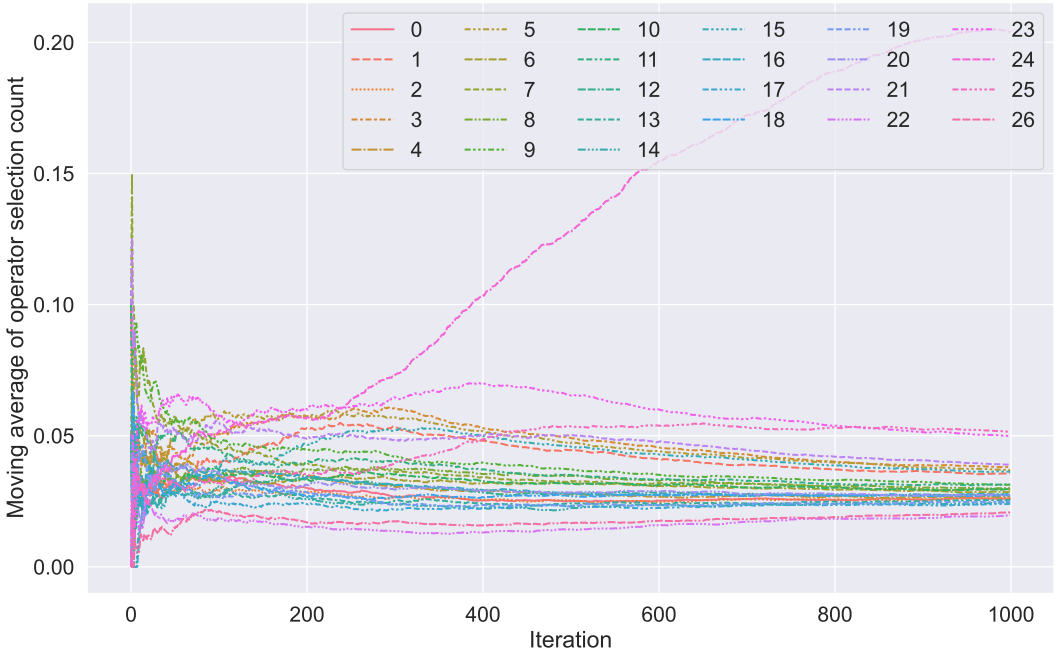


(a) ALNS

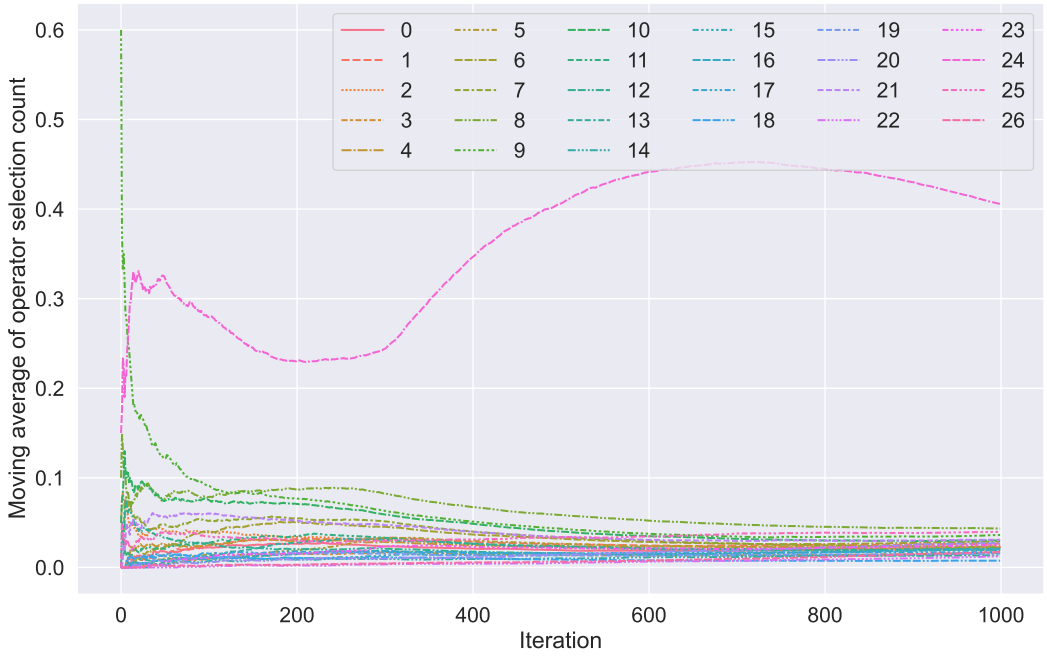


(b) NNALNS

Figure C.25: Moving average of operator selection count for ALNS and NNALNS for V2-d70-e6.

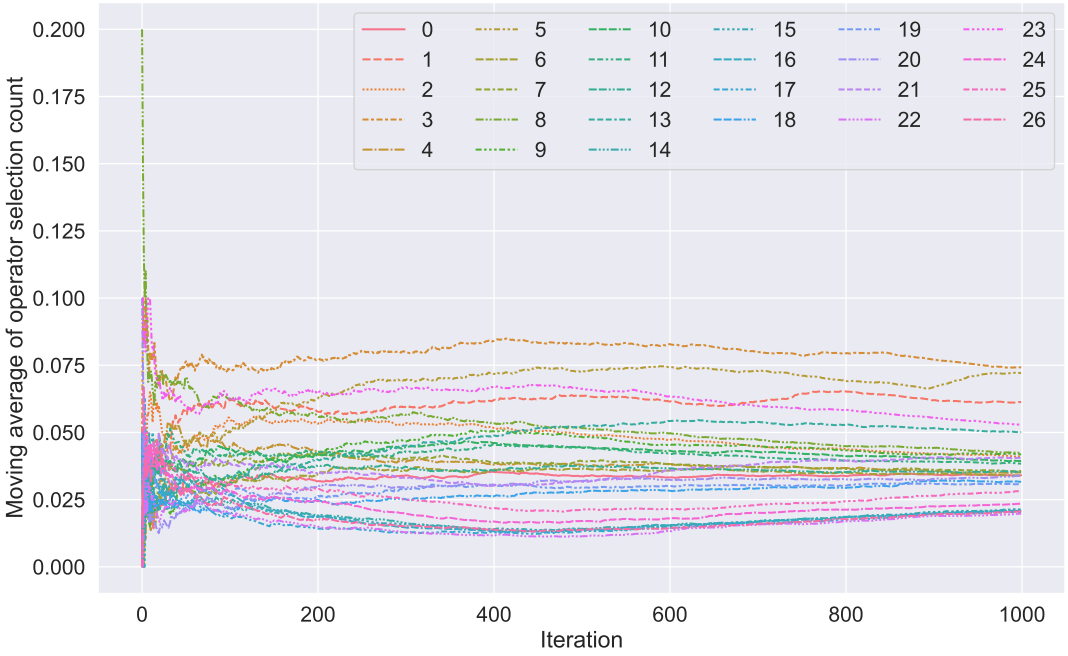


(a) ALNS

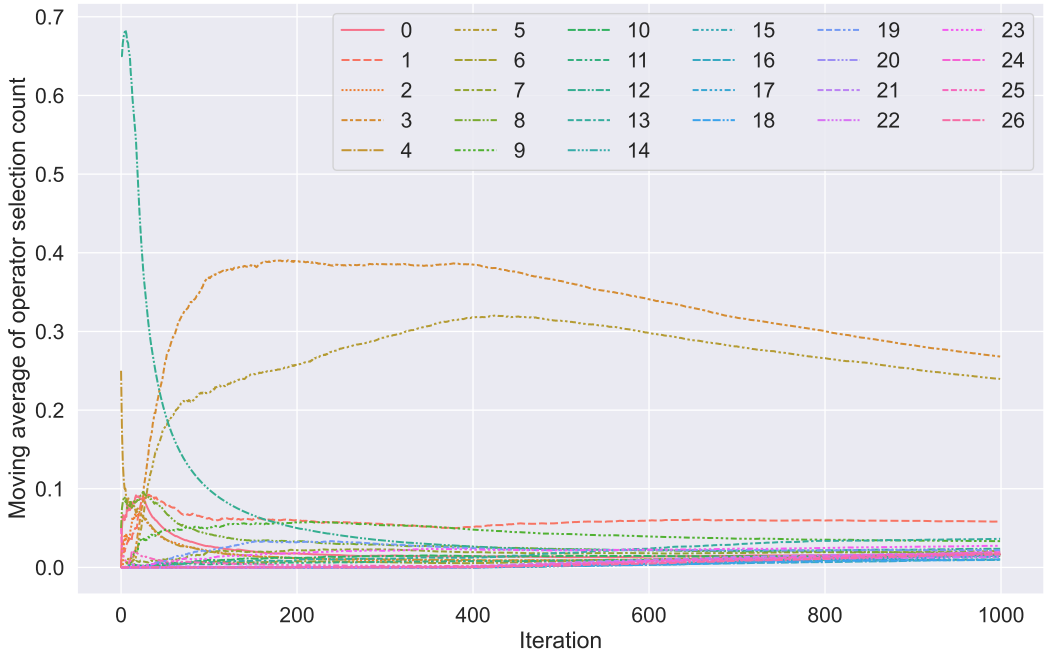


(b) NNALNS

Figure C.26: Moving average of operator selection count for ALNS and NNALNS for V3-d46-e28.

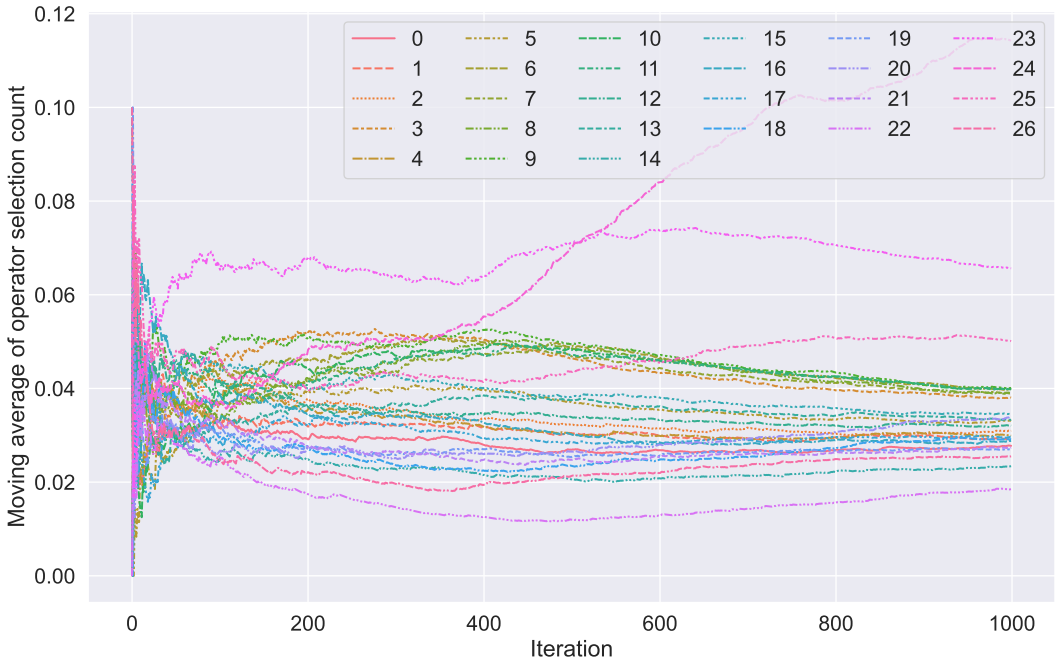


(a) ALNS

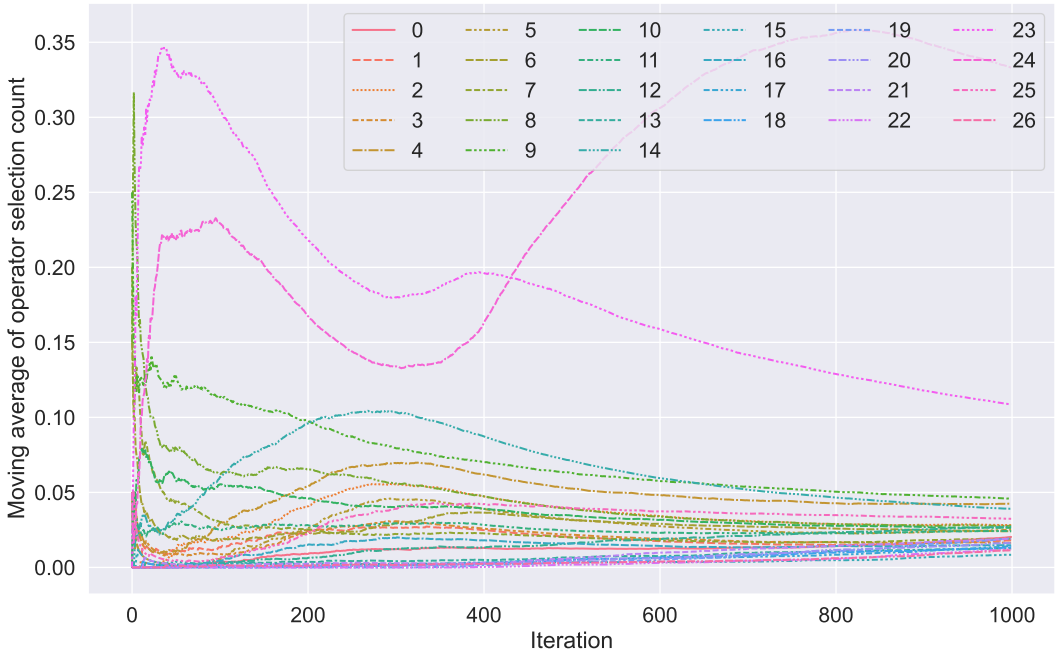


(b) NNALNS

Figure C.27: Moving average of operator selection count for ALNS and NNALNS for V4-d84-e8.

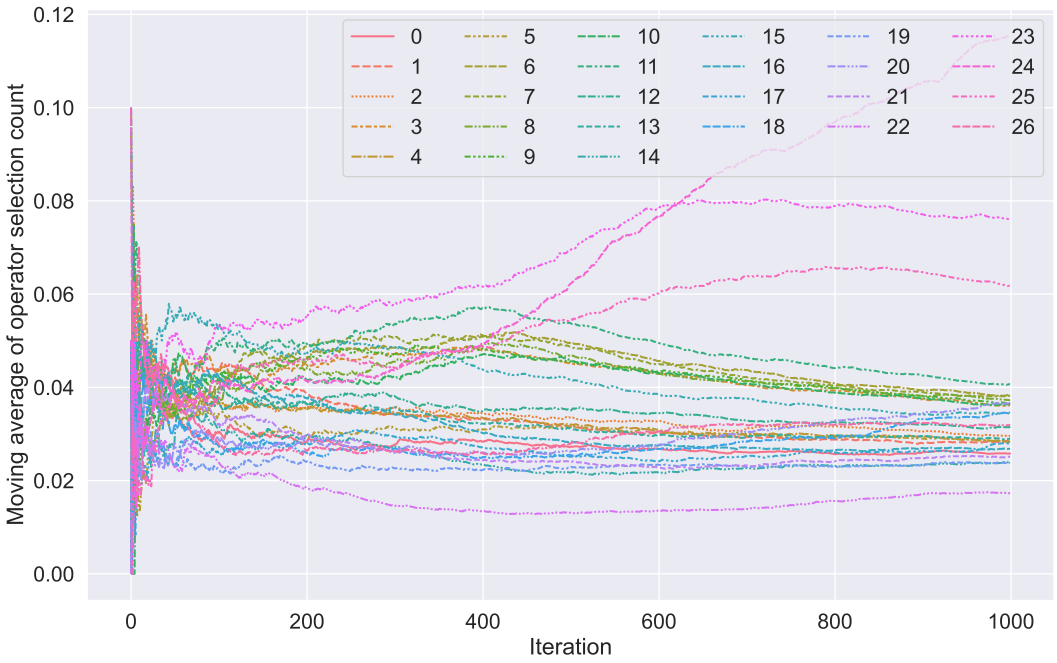


(a) ALNS

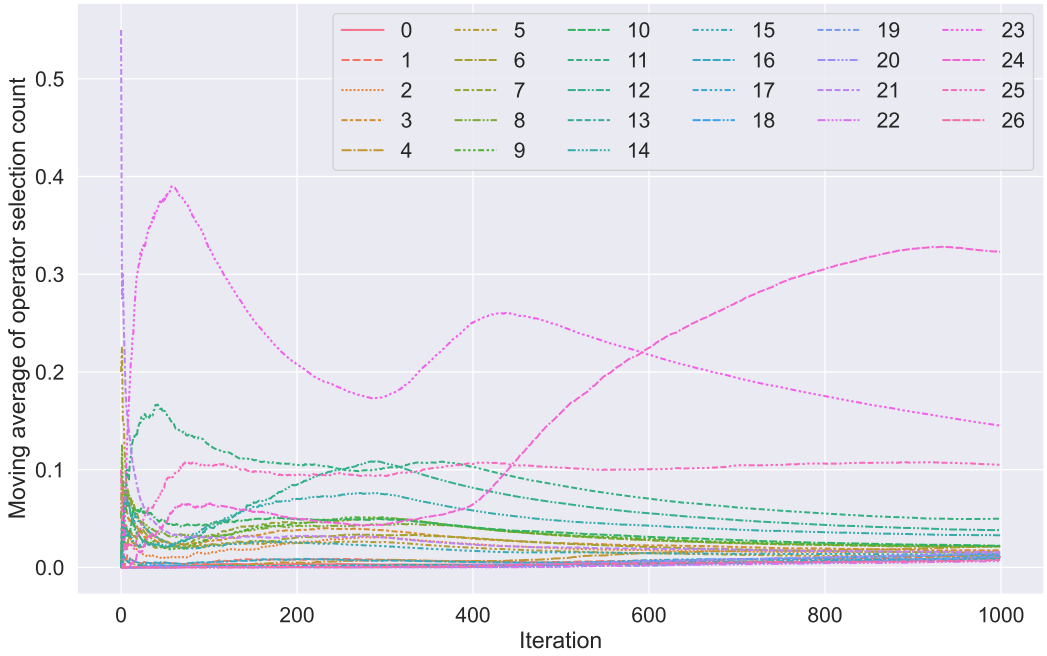


(b) NNALNS

Figure C.28: Moving average of operator selection count for ALNS and NNALNS for V5-d42-e15.



(a) ALNS



(b) NNALNS

Figure C.29: Moving average of operator selection count for ALNS and NNALNS for V6-d98-e15.

C.5 Training

As explained in Section 8.2, we vary between which policy we choose according to the subject of interest during the computational study. When we measure the performance of NNALNS in regards to the objective value, we choose the policy that gave the lowest objective value during training. In contrast, when we compare the reward obtained by NNALNS with and without problem-specific features in Section 8.3.2, we use the last policy.

Figure C.30 and Figure C.31 illustrate the distinction between the two policies. In Figure C.30 we see a downwards trend in the best objective after each update during training. However, after 36 updates, the objective value increases. In Figure C.31 we see that the total reward obtained after each update still increases after 36 episodes, indicating that the network still learns to obtain more rewards.

Based on these observations, it is natural to choose the policy that performed the best according to the objective value during training when we compare the performance of NNALNS in terms of objective value. We choose the last policy for reward comparisons because it enables us to compare the performance after an equal amount of training. As we see in Figure C.31, the total reward shows an upward trend before a slight stagnation. This is a consistent pattern seen during preliminary testing. Thus, choosing the last policy when we compare the performance of NNALNS in terms of reward yields a fair comparison.

Figure C.30: Best objective value development during training on V4-d84-e8.

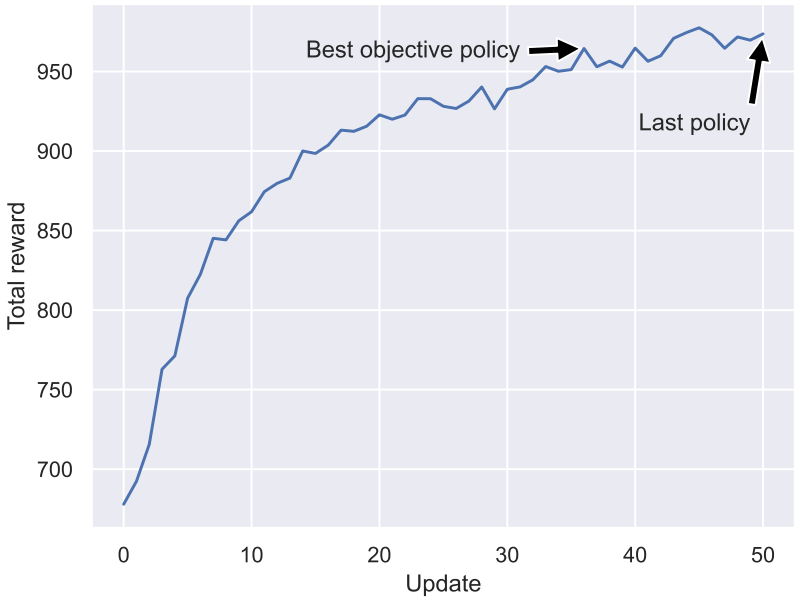
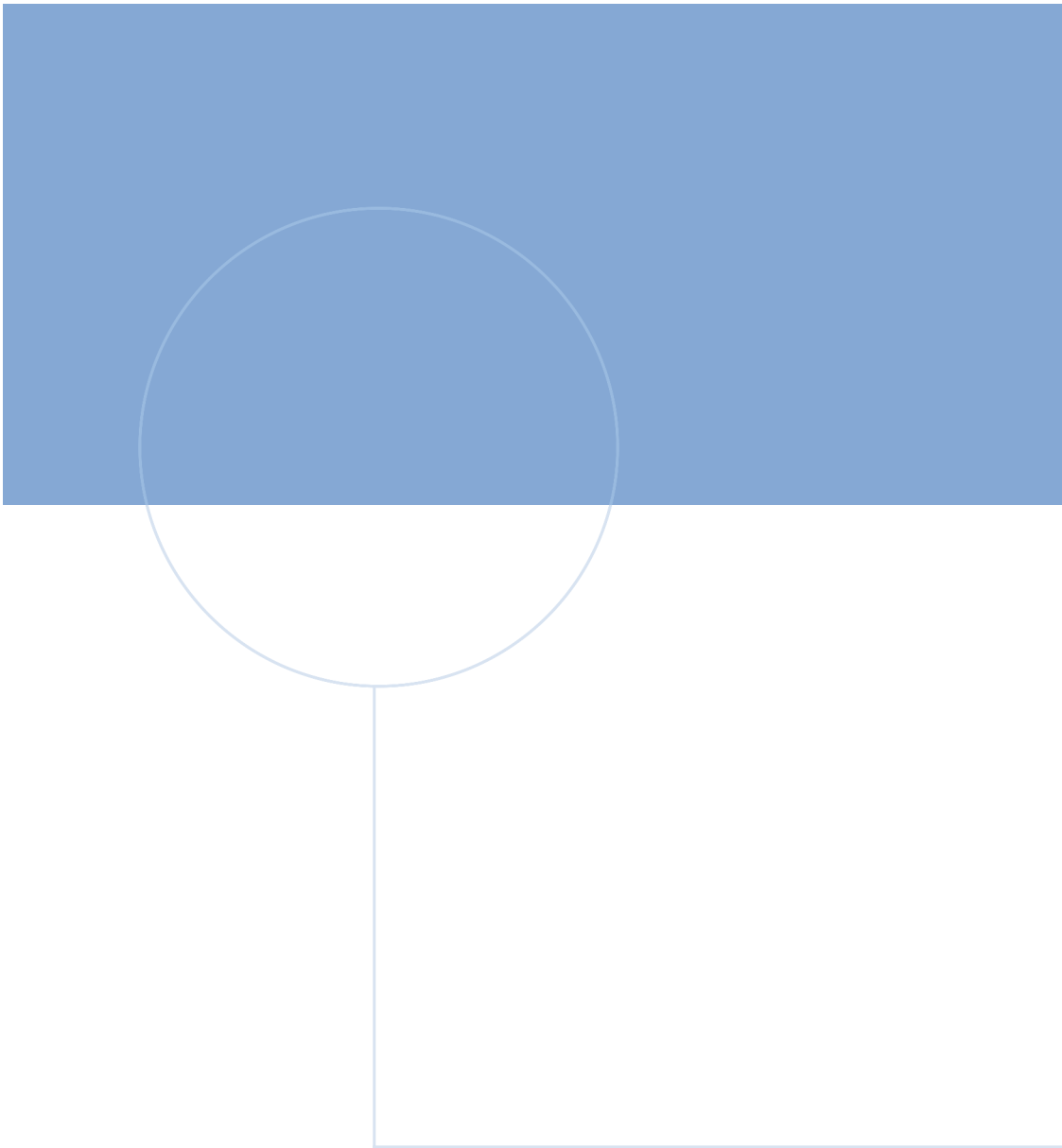


Figure C.31: Total reward development during training on V4-d84-e8.



NTNU

Kunnskap for en bedre verden