A Learning Large Neighborhood Search for the Staff Rerostering Problem*

Fabio F. Oberweger[1], Günther R. Raidl[1], Elina Rönnberg[2], and Marc Huber[1]

[1] Institute of Logic and Computation, TU Wien, Austria
fabio.oberweger@gmail.com, {raidl|mhuber}@ac.tuwien.ac.at
[2] Department of Mathematics, Linköping University, Sweden
elina.ronnberg@liu.se

Abstract. To effectively solve challenging staff rerostering problems, we propose to enhance a large neighborhood search (LNS) with a machine learning guided destroy operator. This operator uses a conditional generative model to identify variables that are promising to select and combines this with the use of a special sampling strategy to make the actual selection. Our model is based on a graph neural network (GNN) and takes a problem-specific graph representation as input. Imitation learning is applied to mimic a time-expensive approach that solves a mixed-integer program (MIP) for finding an optimal destroy set in each iteration. An additional GNN is employed to predict a suitable temperature for the destroy set sampling process. The repair operator is realized by solving a MIP. Our learning LNS outperforms directly solving a MIP with Gurobi and yields improvements compared to a well-performing LNS with a manually designed destroy operator, also when generalizing to schedules with various numbers of employees.

Keywords: Staff rerostering · Large neighborhood search · Imitation Learning · Machine Learning.

1 Introduction

Large neighborhood search (LNS) [33,38] is a powerful meta-heuristics for solving combinatorial optimization problems (COP). It has been successfully applied to many complex problems, including vehicle routing [38], facility location [18], and project scheduling [28]. A common LNS design is to define a neighborhood search by a destroy and repair operator pair that is applied in each iteration [33]. The destroy operator partially destructs the incumbent solution by freeing a subset of the decision variables while fixing the others to their current values. A subproblem is hereby induced, and its space of feasible solutions forms a (large)

* This work is part of the pre-study *Decision support for railway crew planning* supported by KAJT (Capacity in the Railway Traffic System). It is also partially funded by the Doctoral Program *Vienna Graduate School on Computational Optimization*, Austrian Science Foundation (FWF), grant W1260-N35, and the Center for Industrial Information Technology (CENIIT), Project-ID 16.05.

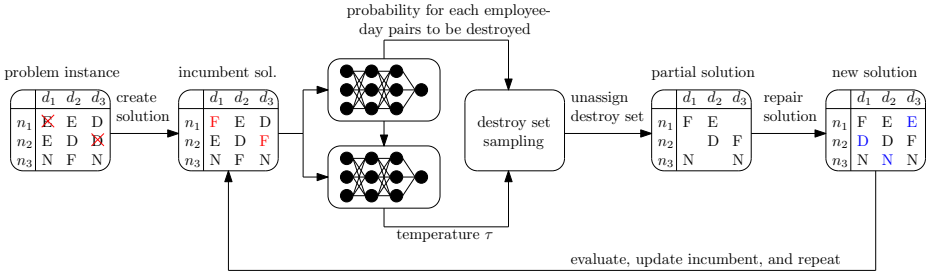

Fig. 1: Principle of the LNS applying trained ML models in the destroy operation.

neighborhood. By exactly or heuristically solving this subproblem, the repair operator tries to improve the previous solution by finding better assignments for these "destroyed" variables. If a new solution with an improved objective value is found, it becomes the new incumbent solution.

The destroy operator has a significant impact on the performance of an LNS. Frequently, a simple random selection of the variables is applied, which, however rarely gives the best results. Manually designing more effective destroy operators often is time-consuming and challenging. Recently, machine learning (ML) based techniques have been suggested to perform this task [2, 14, 39, 40]. These techniques reduce or even eliminate the need for a manual design and have the potential to unveil connections that human experts might not see.

The staff rerostering problem (SRRP) is COP that deals with optimizing and reconstructing work schedules affected by disruptions, e.g., unplanned absences of employees or changes in demand for staff. Inspired by Sonnerat et al. [40], we propose an ML-based LNS consisting of a learning-based destroy operator and the use of a mixed-integer program (MIP) solver as repair method for heuristically solving challenging SRRP instances. Imitation learning is applied to train a conditional generative model predicting weights that indicate which elements of a solution are promising to destroy. Based on these weights, we propose a SRRP-specific sampling strategy for actually choosing the elements to destroy. Moreover, an additional ML model is used to obtain a suitable temperature parameter steering the sampling process. We employ graph neural networks (GNN) utilizing a SRRP-specific graph structure that enables efficient learning and inference for this highly constrained COP. Figure 1 shows our LNS scheme.

Experimental results show that our ML-based LNS outperforms both solving the respective MIP with Gurobi[3] and applying a well-performing LNS with a meaningful manually constructed destroy operator. While our approach is designed specifically for the SRRP, it may be generalized to other problems as well. The components requiring a problem-specific design are the graph structure for the GNN and the destroy set sampling process. Since these may be tailored to the needs of a particular problem, we believe that our approach might perform well on various types of COPs.

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

In Section 2, related work is reviewed and our approach is compared to existing ML-based LNSs. The SRRP is introduced in Section 3 and the LNS framework is discussed in Section 4. Section 5 provides the details of our ML-based destroy operator. Experimental results are presented in Section 6 and concluding comments are given in Section 7. This work is based on the first author's master thesis [31].

2 Related Work

The SRRP is a generalization of the nurse rerostering problem (NRRP). Moz and Pato [25] were the first to formally define the NRRP, which deals with adapting an existing work schedule given employee absences on specific days. While the NRRP only considers employee-based disruptions, e.g., caused by illness, the SRRP also considers changes in demand. Already the NRRP with the single objective of minimizing the differences from the original schedule is NP-hard [27]. The SRRP and the NRRP are both extensions of the classical nurse rostering problem (NRP). For reviews of the NRP, we refer to Ernst et al. [8] and Van den Bergh et al. [5]. The NRRP is not as well studied as the NRP, but has been considered in several works [22, 23, 25–27, 32, 43].

Recently, many researchers have explored the application of ML in combinatorial optimization. ML techniques to learn heuristics for COPs in an end-to-end fashion [1,3,16,19,42] have been steadily improving. Nonetheless, they are usually still outperformed by state-of-the-art classical optimization methods, especially on problems with complex side constraints. For closing this performance gap, a new paradigm often referred to as "learning to search" [39] has evolved. This paradigm generally deals with the usage of ML-based heuristics within other methods. For example, learnable heuristics were used for guiding beam search [17, 30], deciding on how to branch in branch-and-bound (B&B) algorithms [10, 13, 20, 29], and learning destroy or repair operators in LNS [2, 7, 14, 39, 41].

Song et al. [39] proposed an ML-based destroy operator for a decomposition-based LNS solving general MIPs. In this LNS variant, one iteration consists of splitting all variables into disjoint sets and destroying and repairing each variable set one after the other. The authors use reinforcement learning (RL) and imitation learning to train a model performing the variable splits. To obtain data for the imitation learning, they randomly sample multiple decompositions for each solution and take the best. In contrast to the fixed variable subsets in the decomposition-based LNS, Addanki et al. [2] use RL to train a GNN for iteratively selecting one variable at a time until a destroy set of predetermined size is found. A current state in the optimization of a MIP is in [2] modeled by a graph structure called Constraint-Variable Incidence Graph (CVIG). A CVIG consists of one node for each variable and one node for each constraint, and edges represent the occurrence of variables in constraints. Sonnerat et al. [40] avoid the high overhead of one neural network (NN) inference step for each selection of a variable by training a conditional generative model with imitation learning. This model predicts a distribution over all nodes, which they use to sample a destroy

set. The authors also employ a CVIG as input to their GNN. To generate training data, they use a local-branching [9] based mixed-integer programming approach that computes optimal destroy sets. All of the above approaches [2, 39, 40] rely on a MIP solver for repairing solutions.

We build on the approach of Sonnerat et al. [40] but modify it in the following ways. Since CVIGs are huge for the highly constrained SRRPs, they dramatically slow down training and inference already for instances of moderate sizes. Instead, we propose to use a more compact problem-specific graph that efficiently represents the choices to be made by the destroy operator, together with a generalized sampling strategy to choose the destroy set. Moreover, we employ an additional NN predicting suitable values of the temperature parameters for the destroy set sampling process.

3 Staff Rerostering Problem

The SRRP is defined on a set of employees N, a set of days D, and a set of shifts S, which we assume to be an early shift (7 am to 3 pm), the day shift (3 pm to 11 pm), the night shift (11 pm to 7 am), and the free shift modeling off-days. We treat the SRRP as an assignment problem, where each employee $n \in N$ shall be assigned to a shift $s \in S$ on each day $d \in D$. To represent a solution to an SRRP instance, we introduce decision variables $x_{nds} \in \{0, 1\}$, where $x_{nds} = 1$ if and only if employee $n \in N$ is scheduled to work shift $s \in S$ on day $d \in D$. A solution is feasible if it satisfies to following hard constraints:

- Each employee must be assigned to exactly one working shift per day or the free shift.
- An employee has to rest at least 11 hours after each working shift.
- An employee must not be assigned to less than a minimum or more than a maximum number of working shifts in the scheduling period.
- An employee must not be assigned to less than a minimum or more than a maximum number of consecutive working shifts.
- An employee must not have less than a minimum or more than a maximum number of assignments to a shift type in the scheduling period.
- An employee must not have less than a minimum or more than a maximum number of consecutive assignments to a shift type.
- Employees cannot be assigned to a working shift if they are absent for the time of this shift on this day.

Since the SRRP deals with disruptions to an existing schedule, such as absences of employees and changes in the demand for employees, the goal of the SRRP is to comply with the following soft constraints:

- The staffing requirements per day and shift should be met as well as possible.
- The original schedule should be modified as little as possible.

For a detailed formal definition of the SRRP, we refer to [31].

4 Large Neighborhood Search

We now define the repair operator used in each LNS applied in this work and propose a reasonable manually crafted destroy operator that serves as a baseline for comparison in our computational experiments. The construction heuristic (inspired by [35]) used to create an initial solution for the LNS simply takes the provided original schedule and, when an employee is absent on a working shift, changes the assignments to a free shift. The obtained initial solution will therefore typically be infeasible and this has to be considered in the LNS design.

4.1 Random Destroy Operator

Due to the constraints regulating the consecutive number of working shifts and the consecutive assignments per shift type, the repair operator has a greater chance to produce improvements if variables associated with consecutive days are unassigned in the destroy operator. Therefore, our baseline destroy operator randomly selects an employee $n \in N$ and a day $d \in D$ forming an employee-day pair (n, d). In addition, a period $P = \{\max(1, d-z_2), \ldots, d, \ldots, \min(|D|, d+z_2)\}$ is defined containing the days ranging from z_2 days before d to z_2 days after d, respecting that one is the index of the first and $|D|$ the index of the last day. Then, for the selected employee n and day d all variables $x_{nd's}$ for $d' \in P$ are destroyed. This process is repeated z_1 times for each application of the destroy operator; the selection is done without replacement. Both, $z_1 \in \mathbb{N}_0$ and $z_2 \in \mathbb{N}_0$ are fixed strategy parameters. The selection is done without replacement.

4.2 Repair Operator

The repair operator is to apply the Gurobi solver to a MIP representing an SRRP sub-instance induced by the destroy operator. Thus, a given partial solution is repaired by searching for best values for the unassigned variables while the other variables are considered fixed to their current values. As mentioned previously, the construction heuristic may return infeasible solutions. Moreover, the destroy operator might not select all the relevant variables required to turn the solution feasible with one repair operator application. As a consequence, we have to deal with infeasible solutions during the repair operation and the LNS in general. Therefore, an additional MIP is used, where a majority of the hard constraints are transformed into soft constraints. The constraints that are not relaxed are those that ensure that each employee is assigned to exactly one shift per day and that a working shift cannot be assigned to an absent employee. As a result, a solution to this relaxed model can violate the other hard constraints at the cost of additional penalization. The penalization is designed in such a way that infeasible solutions always have a worse objective values than feasible solutions. We refer to [31] for details on the MIP-formulations.

The repair operator uses the MIP with relaxed hard constraints as long as the incumbent solution before the destroy operation was infeasible. When the incumbent solution is feasible, the MIP with the regular hard constraints is

employed. Thus, the two separate MIPs in the repair operator put the focus effectively first on making an infeasible solution feasible and only then to further improve the objective value with respect to the remaining soft constraints.

5 Learning-Based Destroy Operator

The concept of our learning-based destroy operator is to utilize a NN that, given an SRRP instance and a current solution represented by features, returns weights to select promising employee-day pairs $(n, d) \in N \times D$ for which the respective decision variables x_{nds}, $s \in S$, are unassigned, i.e., "destroyed". Let π_θ represents the destroy set model, where θ is the learnable parameters. The model takes a featurized version of a state s^t at step t of an LNS run, consisting of an SRRP instance I and its current solution $x = (x_{nds})_{n \in N,\, d \in D,\, s \in S}$, as an input. The model π_θ outputs a value μ_{nd} for each $(n, d) \in N \times D$ indicating the probability that this employee-day pair is in an optimal destroy set, i.e., a destroy set that when realized yields, after an optimal repair, a solution with a minimum objective value. Note that we consider the size of the destroy set to be fixed to $z_1 \cdot (2z_2 + 1)$ employee-day pairs, just as in the random destroy operator from Section 4.1. More specifically, π_θ consists of two independently trained NNs: one handling states containing infeasible solutions and one dealing with states containing feasible solutions. This distinction is made as we observed that the behavior to learn can be quite different for feasible and infeasible solutions. Also, more training data is created for feasible solutions since the LNS spends more time in the feasible space. By considering two NNs, problems arising from the imbalance in training data are thus avoided. The only difference between the models in π_θ is the data used to train them. Hence, to improve readability, we will only refer to π_θ indicating the respective NN throughout this whole section.

Another important aspect of our learning-based destroy operator is the temperature τ which is a parameter regulating the influence of π_θ's output in the destroy set sampling process. To choose a meaningful τ dynamically in dependence of the progress of the LNS, we use a second model π^{T}_ϕ, which receives a state s^t and the output of π_θ as inputs and predicts a temperature τ for the current situation. Again, π^{T}_ϕ consists of two independently trained NNs, one for infeasible and one for feasible solutions, and for the sake of readability, we will only refer to π^{T}_ϕ.

The next step in our learning-based destroy operator is the destroy set sampling process. Here, we use π_θ's output and the predicted temperature τ to actually select the employee-day pairs to be unassigned in the current solution.

5.1 Markov Decision Process Formulation

A Markov decision process (MDP) [15] is represented by a 4-tuple consisting of the set of states ST, the set of actions A, a transition function T, and a reward function. In our setting, we define a state $s^t \in ST$ at step t of an episode to consist of an SRRP instance I and its current solution x^t. An action $a^t \in \{0, 1\}^{|N \times D|}$ at

step t represents the selection of employee-day pairs to be destroyed. A positive assignment $a^t_{nd} = 1$ for an employee-day pair (n, d) indicates that it is chosen for the destroy set and, for all $s \in S$, the variables x^t_{nds} are unassigned in the current solution. Given a state s^t and an action a^t, the transition function $T : ST \times A \to ST$ determines the next state $s^{t+1} = T(s^t, a^t)$. Here, $T(s^t, a^t)$ is reached by destroying all variables associated with a_t in solution x^t and then repairing the partial solution with the repair operator. We do not define a reward function since it does not play a role in our method.

5.2 Destroy Set Prediction as Conditional Generative Modeling

Inspired by Nair et al. [29], we propose a conditional generative model representing the distribution of actions (i.e., destroy sets) in a current state. For step t in an LNS run, consider a state $s^t \in ST$, an action $a^t \in A$, and the transition function T. Moreover, let $c : ST \to \mathbb{R}$ represent a function returning the objective value of a current solution x^t of state $s^t \in ST$. Define the energy function

$$E(a^t; s^t) = \begin{cases} c(T(s^t, a^t)) & \text{if } c(T(s^t, a^t)) < c(s^t), \\ \infty & \text{otherwise,} \end{cases} \tag{1}$$

over the actions a^t of state s^t, which as in Nair et al. [29] defines the conditional distribution

$$\pi(a^t|s^t) = \frac{e^{-E(a^t;s^t)}}{\sum_{(a')^t} e^{-E((a')^t;s^t)}}. \tag{2}$$

Our learning efforts aim to approximate the conditional distribution in (2) utilizing a model $\pi_\theta(a^t|s^t)$ parameterized by θ. By using the unscaled energy function as presented in (1), destroy sets that leads to solutions with better (lower) objective values after being repaired get a higher probability. Furthermore, destroy sets not leading to improvements in objective value get zero probability. However, since our goal is to generate the best action in each state, we re-scale the energy function such that we assign probability $\pi((a^*)^t|s^t) = 1$ to an optimal action $(a^*)^t$ and a probability of zero to each other action in state s^t. Sonnerat et al. [40] adapted the conditional generative modeling design from Nair et al. [29] in the same way. As a consequence, we only have to consider training data containing optimal actions.

5.3 Sampling Destroy Sets

In Section 4.1, we introduced the baseline random destroy operator, which selects z_1 employee-day pairs $(n, d) \in N \times D$ and destroys all the variables associated with employee-day pairs within the range of z_2 days before to z_2 days after d. Remember that $z_1 \in \mathbb{N}_0$ and $z_2 \in \mathbb{N}_0$ are strategy parameters. Moreover, we described that selecting employee-day pairs without this range does not give good

results since the SRRP contains constraints regarding consecutive working assignments. Although in our learning-based destroy operator, the NN π_θ outputs a value for each employee-day pair describing its probability to be in the destroy set, the same issues remain. Therefore, we propose a new destroy set sampling strategy, where blocks of consecutive employee-day pairs are selected. Each pair $(n, d) \in N \times D$ is assigned an aggregated weight

$$w_{nd} = \sum_{d'=\max(1,\, d-z_2)}^{\min(|D|,\, d+z_2)} (\pi_\theta(a^t_{nd'} = 1 \mid s^t) + \varepsilon)^{\frac{1}{\tau}} \cdot \mathbb{I}[(n, d') \in \mathcal{U}], \tag{3}$$

where τ a temperature parameter to strengthen or weaken the influence of the NN, $\varepsilon > 0$ is an offset to give every pair a non-zero weight, and $\mathcal{U}$ is the destroy set selected so far. These weights can be interpreted as the sum of all NN outputs for employee n in a window of $2z_2 + 1$ consecutive days around day d, specifying the importance to add this range of employee-day pairs to the destroy set. We randomly select an employee-day pair (n, d) proportional to these weights w_{nd}, add pairs (n, d') for all $d' \in \{\max(1, d - z_2), \ldots, d, \ldots, \min(|D|, d + z_2)\}$ to $\mathcal{U}$, and repeat this process z_1 times. Eventually, for each $(n, d) \in \mathcal{U}$, we unassign variables $x^t_{nd's}$ for each shift $s \in S$ in the current solution.

5.4 Neural Networks

A GNN [12, 36] is a NN architecture taking a graph as an input. It is typically independent of a specific graph size and able to represent underlying structural properties of a given graph by mapping it to a graph embedding expressed as vectors of values on the graph's nodes. We propose the following custom graph representation for the SRRP, which is not a reformulation of the SRRP as a graph problem but reflects a knowledge graph containing information for choosing suitable employee-day pairs. We define this graph as $G = (V, E, X)$, where V is the set of nodes, E the set of edges, and $X \in \mathbb{R}^{|V| \times p}$ a node feature matrix assigning each node $v \in V$ a p-dimensional feature vector $X_v = (X_{v,1} \ldots X_{v,p})$.

The set of nodes $V = V_{\text{emp}} \cup V_{\text{assign}} \cup V_{\text{day}}$ is composed of three different types which are employee $V_{\text{emp}} = \{n \mid n \in N\}$, assignment $V_{\text{assign}} = \{(n, d) \mid n \in N, d \in D\}$, and day $V_{\text{day}} = \{d \mid d \in D\}$ nodes. The assignment nodes represent the employee-day pairs. There are edges between an assignment node and its associated employee and day nodes. Moreover, since the days represent a sequence, we add edges between consecutive days.

There are thus $O(|N| \cdot |D|)$ nodes and edges, which is substantially less than in a CVIG for the underlying MIP (e.g., as used in [2]). A reasonably fast training and inference can therefore be expected. The interpretation of this representation is that an employee is involved in an assignment and this assignment takes place on a specific day in the planning horizon. A state $s^t \in ST$, consisting of an SRRP instance and its current solution x^t, holds all the required information to create such a graph structure. If we later use a state directly as an input to our NN, we implicitly assume that it is first transformed into such a graph.

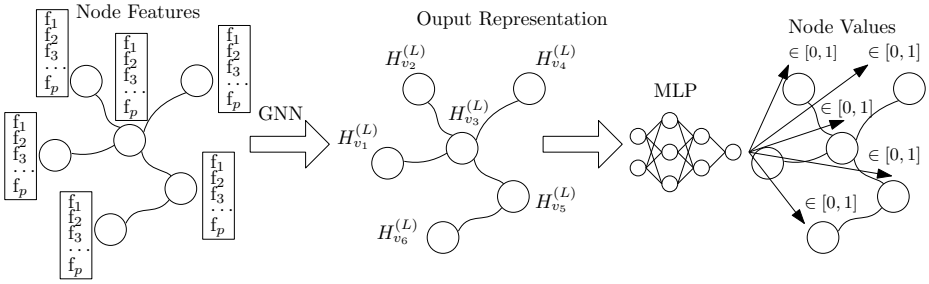

Fig. 2: Simplified representation of the NN architecture for the destroy set model π_θ; figure inspired by [6].

Naturally, we have different kinds of features with respect to employees, days, and assignments. For example, employee node features include the number of assignments to each shift $s \in S$ of an employee, day node features include the total number of assignments to each shift $s \in S$ on a given day, and assignment node features include the currently and originally assigned shift for an employee-day pair. For the complete list of the features used, we refer to [31]. To provide all these features to the GNN, one possibility would be to associate the individual features with the respective types of nodes in the graph representation and to apply the relational graph convolutional network (R-GCN) [37] for handling such an inhomogeneous graph with different feature types on different node types. However, similar to Chalumeau et al. [6], we use a simpler approach enabling us to employ more general GNNs for homogeneous graphs. Let $\mathrm{f}_1^{\mathrm{emp}}, \ldots, \mathrm{f}_{q^{\mathrm{emp}}}^{\mathrm{emp}}$ be the employee features, $\mathrm{f}_1^{\mathrm{assign}}, \ldots, \mathrm{f}_{q^{\mathrm{assign}}}^{\mathrm{assign}}$ the assignment features, and $\mathrm{f}_1^{\mathrm{day}}, \ldots, \mathrm{f}_{q^{\mathrm{day}}}^{\mathrm{day}}$ the day node features, where $q^{\mathrm{emp}}, q^{\mathrm{assign}}, q^{\mathrm{day}}$ are the number of employee, assignment, and day node features, respectively. Then each feature vector x_v of a node v, independent of the node type, is of the form

$$(\mathrm{f}_1^{\mathrm{emp}}, \ldots, \mathrm{f}_{q^{\mathrm{emp}}}^{\mathrm{emp}}, \mathrm{f}_1^{\mathrm{assign}}, \ldots, \mathrm{f}_{q^{\mathrm{assign}}}^{\mathrm{assign}}, \mathrm{f}_1^{\mathrm{day}}, \ldots, \mathrm{f}_{q^{\mathrm{day}}}^{\mathrm{day}}, \mathrm{f}_1^{\mathrm{enc}}, \mathrm{f}_2^{\mathrm{enc}}, \mathrm{f}_3^{\mathrm{enc}}) \tag{4}$$

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

Architectures. So far, we have established the underlying graph structure. Figure 2 shows a simplified representation of the NN architecture of π_θ. First, we employ a GNN similar to the neural network for graphs from [24]. Our GNN updates the feature representation $H^{(l)}$ in a layer l by applying the update function

$$H^{(l)} = \sigma\left(H^{(l-1)} W_1^{(l)} + A H^{(l-1)} W_2^{(l)} + b^{(l)}\right), \tag{5}$$

where $H^{(0)} = X$, A is the adjacency matrix, σ is a non-linear activation function, $b^{(l)} \in \mathbb{R}^q$ is the learnable bias, and $W_1^{(l)}, W_2^{(l)} \in \mathbb{R}^{m \times q}$ denote the weight matrices for layer l, where $m, q \in \mathbb{N}_0$ are the input and output feature dimensions, respectively. Applying the GNN to an input yields the node embedding $H^{(L)}$ of the graph, where $H_v^{(L)}$ is a vector for each node $v \in V$. These vectors from the GNN are further processed by a traditional multi-layer perceptron (MLP) utilizing a sigmoid activation in the last layer to yield a value in [0,1] for each node. We only require the final output for the assignment nodes.

To make the connection to our conditional generative modeling approach and the conditional distribution π from Eq. (2) in Section 5.2, let π_θ be our previously presented NN, where θ are all the learnable parameters, including the GNN and MLP weights. Remember that an action a_{nd}^t at step t indicates whether an employee-day pair $(n, d) \in N \times D$ is contained in the destroy set ($a_{nd}^t = 1$) or not ($a_{nd}^t = 0$). Also, remember that the sets V_{assign} and $N \times D$ are isomorphic, meaning that there is a one-to-one correspondence between each assignment node and employee day pair $(n, d) = v \in V_{\text{assign}}$. As Nair et al. [29] and Sonnerat et al. [40], we define π_θ to be a conditional-independent model of the form

$$\pi_\theta(a^t \mid s^t) = \prod_{(n,d) \in V_{\text{assign}}} \pi_\theta(a_{nd}^t \mid s^t), \tag{6}$$

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

$$t_{nd} = \texttt{MLP}(H_{(n,d)}^{(L)}; \theta), \tag{7}$$

$$\mu_{nd} = \pi_\theta(a_{nd}^t = 1 \mid s^t) = \frac{1}{1 + e^{-t_{nd}}}. \tag{8}$$

As pointed out by Nair et al. [29], it is not possible to accurately model a multimodal distribution using the assumption of conditional independence. Despite that, they reported strong empirical results. Mathematically more accurate alternatives are autoregressive models [4] or inferring one employee-day pair at a time by repeatedly evaluating the NN. However, this increased accuracy comes at the cost of substantially slower inference times [29,40], which are not reasonable in our setting.

The architecture of the temperature NN $\pi_\phi^\intercal$ is similar to the architecture of the destroy set model π_θ but there are some key differences. The temperature model $\pi_\phi^\intercal$ shall predict how strongly π_θ shall influence the destroy set sampling process in a specific state s^t. Therefore, we add the output of π_θ as an additional feature to the node features of the graph representation. More specifically, we append the common node feature vector with $\mu_{nd} = \pi_\theta(a_{nd}^t = 1 \mid s^t)$ for assignment nodes $(n, d) = v \in V_{\text{assign}}$ and zero for every other node $v \in V_{\text{emp}} \cup V_{\text{day}}$. Another difference is that a read-out layer is applied on the final node representations

$H_v^{\intercal,(L)}$. This read-out layer aggregates the information over all nodes into a single vector by applying $\sum_{v \in V} H_v^{\intercal,(L)}$. Finally, we employ an MLP with a softmax function in the last layer on this vector to return a probability for each temperature in a predefined set $\mathcal{T} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 5\}$ to be the best selection. This classification-based approach turned out to be more effective in practice than a regression model.

5.5 Training

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

$$\mathcal{L}(\theta) = -\sum_{j=1}^{M} \sum_{t=1}^{T_j} \log \pi_\theta(a_{(j)}^t \mid s_{(j)}^t), \tag{9}$$

which is the negative log likelihood of the expert actions.

The training set $\mathcal{D}_{\text{train}}^{\intercal} = \{(s_{(j)}^{1:T_j}, o_{(j)}^{1:T_j}, y_{(j)}^{1:T_j})\}_{j=1}^{M^{\intercal}}$ for the temperature model $\pi_\phi^{\intercal}$ contains the outputs $\{o_{(j)}^t\}_{t=1}^{T_j}$ of π_θ in the respective states $\{s_{(j)}^t\}_{t=1}^{T_j}$ for each sampled trajectory $j = 1, \dots, M^{\intercal}$. The associated labels $\{y_{(j)}^t\}_{t=1}^{T_j}$ consist of a one-hot encoding of the temperature found to be best by the expert for each time step t of a trajectory j. Eventually, we optimize the weights ϕ by minimizing the cross-entropy loss

$$\mathcal{L}^{\intercal}(\phi) = -\sum_{j=1}^{M^{\intercal}} \sum_{t=1}^{T_j} y_{(j)}^t \log \pi_\phi^{\intercal}(s_{(j)}^t, o_{(j)}^t). \tag{10}$$

We perform each training in mini-batches of size 32. In addition to $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{train}}^{\intercal}$, we also create validation sets of the same form as the respective training sets containing about a fourth of the total generated trajectories. This data is hold out from $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{train}}^{\intercal}$ to evaluate the progress on unseen data during training. Furthermore, we apply early stopping [11, p. 246] to avoid overfitting. As optimizer, we use ADAM [21] with a learning rate of 0.001 and an exponential decay rate of 0.9 for the first and 0.999 for the second momentum.

5.6 Training Data Generation

Our data generation process is inspired by the expert policy from Sonnerat et al. [40], which uses local branching [9] to create optimal destroy sets in a given state. In local branching, a constraint is added to the MIP that allows at most a certain number of decision variables to change compared to a given incumbent solution. In the following, we refer to the MIP extended with such a local branching constraints as extended or local branching-based MIP. If this

extended MIP is solved to optimality in a current state s^t, an optimal destroy set can be derived by comparing the old solution x^t with the new solution x^{t+1} and collecting the variables with changed values. Since our destroy sets do not directly consist of decision variables but employee-day pairs, the local branching constraint is in our case

$$\sum_{(n,d,s)\in N\times D\times S:\, x^t_{nds}=0} x^{t+1}_{nds} + \sum_{(n,d,s)\in N\times D\times S:\, x^t_{nds}=1} (1 - x^{t+1}_{nds}) \;\leq\; 2\eta, \qquad (11)$$

which ensures that at most η employee-day pairs change. Note that one employee-day pair change always implies the change of two x_{nds} variables, since each employee must be assigned to exactly one shift, including the free shift, on each day. In the following, we refer to iteratively solving the local branching-based MIP and extracting the associated destroy set as our expert policy π^*.

To generate training samples for the destroy set model π_θ, we apply the Dataset Aggregation (DAGGER) algorithm which is an extension of classical behavior cloning [34]. In the first iteration, the expert policy is used to sample trajectories for training instances. Sampling a trajectory using a policy $\hat{\pi}$ means that in each state s^t at step t of an episode (LNS run), we store state s^t and the expert action $\pi^*(s^t)$ as a tuple in a dataset $\mathcal{D}$, use $\hat{\pi}$ to create a destroy set a^t, and move to the next state $s^{t+1} = T(s^t, a^t)$. Then, a model $\hat{\pi}_1$ is trained on the expert actions for all the encountered states in $\mathcal{D}$, to mimic the expert policy. In the second iteration, we use $\hat{\pi}_1$ to sample more trajectories and add more data to $\mathcal{D}$. For this learned policy $\hat{\pi}_1$, we apply the destroy set sampling strategy from Section 5.3 with temperature $\tau = 1$ to generate a destroy set a^t in a current state s^t. This process is in general iterated a certain number of times. Eventually, $\mathcal{D}$ is the final dataset which is used to train the destroy set model π_θ.

The idea behind this algorithm is that trained models may encounter very different states than the expert strategy and it is also important to train on those. Preliminary results showed in our case that DAGGER iterations slightly improve the result, although we ultimately decided to only perform two iterations as our expert policy is very time-consuming due to the iterative solving of the extended MIP. To further speed up the data generation, we took the following additional measures. First, we terminate the solving of the local branching-based MIP after a defined time limit, yielding a destroy set that is not guaranteed to be optimal. Second, we execute the trajectory sampling for each SRRP training instance in parallel. Lastly, we only create a training sample for every third visited state when sampling trajectories with a trained model if the solution is feasible. See [31] for more details on these refinements.

Concerning the temperature model π^{T}_ϕ, that steer the diversity of the randomized employee-day pair selection process by choosing a value of τ. A lower value increases the impact of π_θ, while a higher one decreases it. We may also interpret π^{T}_ϕ as an evaluator of π_θ. If the output of π_θ for a specific state s^t is highly promising/reliable, a lower temperature should be predicted for τ to increase the influence of π_θ. Vice versa, higher temperatures should be predicted if the current output of π_θ is not so likely to yield an improvement of the incumbent solution. To collect training data for π^{T}_ϕ, we produce multiple trajectories

using π_θ, where each temperature $\tau \in \mathcal{T}$ is applied to sample three destroy sets in a state s^t at step t of an episode. Let τ_t^* be the temperature giving the best destroy sets on average in a state s^t. The best average performance is determined as follows: If the solution has been improved, we consider the objective value of the newly created solution, otherwise the objective value of the initially constructed solution is used. At each step t, we add s^t, $\pi_\theta(s^t)$, and τ_t^* to a dataset $\mathcal{D}^\intercal$. Then, we move to the next state $T(s^t, a^t_{\tau_t^*})$ using an action sampled with τ_t^* and repeat this process until the termination of the LNS.

6 Experimental Evaluation

All algorithms were implemented in Julia[4] 1.6.1 and all MIPs were solved by Gurobi 9.1.0. The experiments were executed in single-threaded mode on a machine with an Intel Xeon E5–2640 processor with 2.40 GHz and a memory limit of 16 GB. Since it is essential to find high-quality solutions fast in practice, we worked with a time limit of 900 seconds to evaluate our optimization algorithms. We use %-gap, an optimality gap in percent, to evaluate computational performance. For an objective value o, the value of %-gap is $100 \cdot (o - lb)/o$, where lb is a lower bound for the optimal value obtained by solving the original MIP with Gurobi for three hours.

A time limit of five seconds is applied for solving the sub-MIP in a call of the repair operator of each LNS. For both the random and the learning-based destroy operator, the values $z_1 = 150$ and $z_2 = 2$ are used since they gave a good performance in preliminary experiments with the random destroy operator.

As baselines for our computational comparisons, we solved the MIP with Gurobi and used the LNS with the random destroy operator. In the following figures and tables, we will refer to these approaches as ILP and LNS_RND, respectively, while LNS_NN denotes the ML-based LNS. As the training data generation is time-consuming, we trained the NNs of LNS_NN on data generated from a random single schedule with $|N| = 110$ employees and various sets of disruptions only. In total, we used 200 and 150 random sets of disruptions for this schedule to generate training trajectories for the destroy set and the temperature model, respectively. To accelerate the training data generation, a time limit of 30 minutes and $\eta = 375$ was used for solving the extended MIP.

In the following experiment, we aim to show the strong performance of LNS_NN and its ability to generalize to different schedules with different numbers of employees despite the rather restricted training data generation. For testing, we randomly created four schedules with 120, 130, 140, and 150 employees. For each of these schedules, 30 random sets of disruptions are sampled such that there is a total of 120 SRRP instances. The time horizon is four weeks, i.e., $|D| = 28$. The instance generation and training process are detailed in [31].

Table 1 presents the results including average %-gaps and respective standard deviations, average objective values, average numbers of performed iterations, grouped according to the instance size $|N|$. It is clear that LNS_NN performs

[4] https://julialang.org

Table 1: Average results of the trained LNS_NN, the LNS_RND, and the ILP. For each number of employees $|N|$, there are 30 SRRP instances. Objective values are divided by 10^3 for improved readability.

	LNS_NN			LNS_RND			ILP	
$\|N\|$	%-gap	obj. val.	iter.	%-gap	obj. val.	iter.	%-gap	obj. val.
120	**3.42** $\pm$ 0.53	**926**	118.93	4.36 $\pm$ 0.57	935	127.27	34.18 $\pm$ 13.74	1,418
130	**4.45** $\pm$ 0.40	**1,079**	113.30	5.22 $\pm$ 0.51	1,088	128.10	28.51 $\pm$ 13.48	1,495
140	**8.04** $\pm$ 0.60	**1,314**	99.77	9.20 $\pm$ 0.58	1,331	105.43	31.31 $\pm$ 11.79	1,817
150	**5.74** $\pm$ 0.41	**1,427**	95.57	6.87 $\pm$ 0.46	1,446	100.97	26.71 $\pm$ 11.30	1,880

better than LNS_RND for each of these instance classes even if it is trained only on the main schedule with $|N| = 110$. On average, LNS_NN improves the gap of LNS_RND by about one percent for each instance class. In total, LNS_RND only reached better solutions than LNS_NN for two of the 120 instances from this experiment. LNS_NN performs less iterations than LNS_RND. This decrease in iterations primarily comes from the fact that LNS_NN finds more meaningful destroy sets requiring more time to be repaired. These outcomes are even more substantial considering that already LNS_RND is a well-performing approach, which clearly surpassing the pure MIP solving. Considering optimality gap on average, LNS_NN outperforms ILP by factors between 3.89 and 9.99 across all instance groups.

Figure 3a visualizes the dominance of LNS_NN over LNS_RND and the significance of the differences with boxplots. Finally, Figure 3b shows how the solution quality develops over time for LNS_NN and LNS_RND, aggregated over all runs for all benchmark instances. Most important to note is that in the beginning of the search, LNCS_NN finds improving solutions much faster than LNS_RND does. While LNS_NN reaches an optimality gap of less than 10% after 271 seconds on average, it takes LNS_RND 420 seconds. Also, LNS_NN finds feasible solutions after approximately 190 seconds and LNS_RND only after 310 seconds on average. Last but not least, remarkable are also the small standard deviations shown by the shaded areas, indicating the robustness of both LNS variants.

7 Conclusion and Future Work

We proposed a learning LNS for solving the SRRP, where the destroy operator is guided by a ML model trained upfront on representative problem instances with an imitation learning approach. More specifically, a conditional generative model predicts weights for all employee-day pairs, which are used in a randomized sampling strategy based on consecutive day selection to derive high-quality destroy sets. We use a custom graph structure modeling a current solution to the SRRP as an input to a GNN. Other so far proposed learning LNS approaches employ CVIGs, resulting in prohibitively huge graphs for problems like the SRRP. In addition to the main NN predicting the employee-day pair weights, a second NN

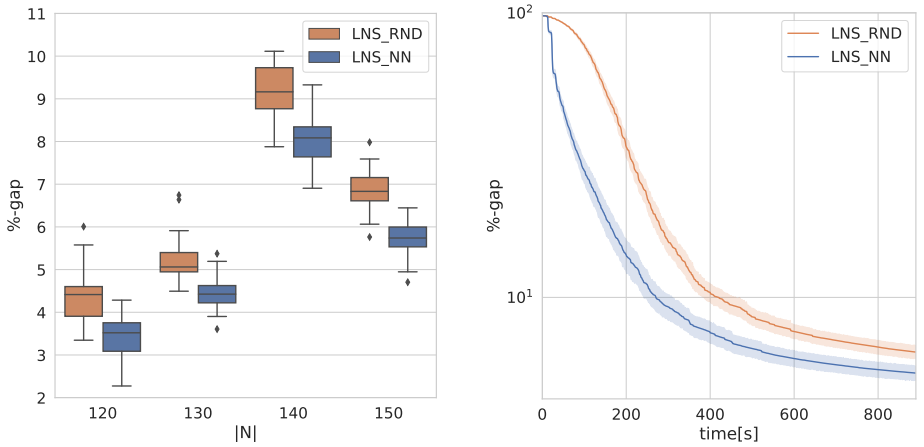

Fig. 3: (a) Boxplots for the %-gaps finally obtained by LNS_RND and LNS_NN in dependence of the numbers of employees $|N|$. (b) Solution quality development over time, aggregated over all runs/instances; note that %-gaps are here presented in log-scale.

predicts a temperature value that controls the diversity of the destroy set sampling process. Experimental results clearly indicate the benefits of the learning LNS over a reasonably designed and already well working classical LNS in terms of solution quality and speed of solution improvement. Noteworthy also is the excellent generalization capability to unseen and even larger SRRP instances. The proposed learning LNS provides a scheme that may also be promising for other classes of highly constrained COPs.

In future work, training and testing of the approach on more diverse and even larger SRRP instances would be interesting. Investigating different variants of GNNs may lead to further improvements. Finally, finding ways to apply RL to optimize a performance metric directly instead of relying on the time-expensive imitation learning is a promising research direction.

References

1. Abe, K., Xu, Z., Sato, I., Sugiyama, M.: Solving NP-hard problems on graphs with extended AlphaGo Zero. arXiv preprint arXiv:1905.11623 (2020)
2. Addanki, R., Nair, V., Alizadeh, M.: Neural large neighborhood search. In: Learning Meets Combinatorial Algorithms at Conf. on Neural Information Processing Systems (2020)
3. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. In: Workshop Proc. of the 5th Int. Conf. on Learning Representations. OpenReview.net (2017)

4. Bengio, Y., Bengio, S.: Modeling high-dimensional discrete data with multi-layer neural networks. In: Advances in Neural Information Processing Systems. vol. 12, pp. 400–406. MIT Press (1999)
5. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**, 367–385 (2013)
6. Chalumeau, F., Coulon, I., Cappart, Q., Rousseau, L.M.: Seapearl: A constraint programming solver guided by reinforcement learning. In: Proc. of the 18th Int. Conf. on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR). LNCS, vol. 18, pp. 392–409. Springer (2021)
7. Chen, M., Gao, L., Chen, Q., Liu, Z.: Dynamic partial removal: A neural network heuristic for large neighborhood search. arXiv preprint arXiv:2005.09330 (2020)
8. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research **153**, 3–27 (2004)
9. Fischetti, M., Lodi, A.: Local branching. Mathematical programming **98**, 23–47 (2003)
10. Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. In: Advances in Neural Information Processing Systems. vol. 32, pp. 15554–15566. Curran Associates, Inc. (2019)
11. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
12. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Proc. of the IEEE Int. Joint Conf. on Neural Networks 2005. vol. 2, pp. 729–734. IEEE (2005)
13. He, H., Daume III, H., Eisner, J.M.: Learning to search in branch and bound algorithms. Advances in Neural Information Processing Systems **27**, 3293–3301 (2014)
14. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. In: Proc. of the 24th European Conf. on Artificial Intelligence. FAIA, vol. 325, pp. 443–450. IOS Press (2020)
15. Howard, R.A.: Dynamic programming and Markov processes. John Wiley (1960)
16. Huang, J., Patwary, M., Diamos, G.: Coloring big graphs with alphago zero. arXiv preprint arXiv:1902.10162 (2019)
17. Huber, M., Raidl, G.R.: Learning beam search: Utilizing machine learning to guide beam search for solving combinatorial optimization problems. In: Machine Learning, Optimization, and Data Science – 7th International Conference, LOD 2021. LNCS, vol. 11943. Springer (2021), to appear
18. Jatschka, T., Oberweger, F.F., Rodemann, T., Raidl, G.R.: Distributing battery swapping stations for electric scooters in an urban area. In: Proc. of the 11th Int. Conf. on Optimization and Applications. LNCS, vol. 12422, pp. 150–165. Springer (2020)
19. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems. vol. 30, pp. 6348–6358. Curran Associates, Inc. (2017)
20. Khalil, E.B., Bodic, P.L., Song, L., Nemhauser, G.L., Dilkina, B.N.: Learning to branch in mixed integer programming. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence. pp. 724–731. AAAI Press (2016)
21. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Proc. of the 3rd Int. Conf. on Learning Representations (2015)

22. Maenhout, B., Vanhoucke, M.: An evolutionary approach for the nurse rerostering problem. Computers & Operations Research **38**, 1400–1411 (2011)
23. Maenhout, B., Vanhoucke, M.: Reconstructing nurse schedules: Computational insights in the problem size parameters. Omega **41**, 903–918 (2013)
24. Micheli, A.: Neural network for graphs: A contextual constructive approach. IEEE Transactions on Neural Networks **20**, 498–511 (2009)
25. Moz, M., Pato, M.V.: An integer multicommodity flow model applied to the rerostering of nurse schedules. Annals of Operations Research **119**, 285–301 (2003)
26. Moz, M., Pato, M.V.: Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. Annals of Operations Research **128**, 179–197 (2004)
27. Moz, M., Pato, M.V.: A genetic algorithm approach to a nurse rerostering problem. Computers & Operations Research **34**, 667–691 (2007)
28. Muller, L.F.: An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In: Proc. of the VIII Metaheuristics International Conference 2009 (2009)
29. Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., et al.: Solving mixed integer programs using neural networks. arXiv preprint arXiv:2012.13349 (2020)
30. Negrinho, R., Gormley, M.R., Gordon, G.J.: Learning beam search policies via imitation learning. In: Advances in Neural Information Processing Systems. vol. 31, p. 10675–10684. Curran Associates Inc. (2018)
31. Oberweger, F.F.: A Learning Large Neighborhood Search for the Staff Rerostering Problem. Diploma thesis, Institute of Logic and Computation, TU Wien, Austria (2021)
32. Pato, M.V., Moz, M.: Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic. Journal of Heuristics **14**, 359–374 (2008)
33. Pisinger, D., Ropke, S.: Large neighborhood search. In: Handbook of metaheuristics, pp. 399–419. Springer (2010)
34. Pomerleau, D.A.: Alvinn: An autonomous land vehicle in a neural network. In: Advances in Neural Information Processing Systems. vol. 1, p. 305–313. MIT Press (1988)
35. Rönnberg, E., Larsson, T., Bertilsson, A.: Automatic scheduling of nurses: What does it take in practice? In: Pardalos, P., Georgiev, P., P., P. (eds.) Systems Analysis Tools for Better Healthcare Delivery, pp. 151–178. Springer (2012)
36. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**, 61–80 (2008)
37. Schlichtkrull, M., Kipf, T.N., Bloem, P., Van Den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: Proc. of the 15th Int. Conf. on the Semantic Web. LNCS, vol. 10843, pp. 593–607. Springer (2018)
38. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Proc. of the 4th Int. Conf. on Principles and Practice of Constraint Programming. LNCS, vol. 1520, pp. 417–431. Springer (1998)
39. Song, J., Lanka, R., Yue, Y., Dilkina, B.: A general large neighborhood search framework for solving integer linear programs. In: Advances in Neural Information Processing Systems. vol. 33, pp. 20012–20023. Curran Associates, Inc. (2020)
40. Sonnerat, N., Wang, P., Ktena, I., Bartunov, S., Nair, V.: Learning a large neighborhood search algorithm for mixed integer programs. arXiv preprint arXiv:2107.10201 (2021)

41. Syed, A.A., Akhnoukh, K., Kaltenhaeuser, B., Bogenberger, K.: Neural network based large neighborhood search algorithm for ride hailing services. In: Proc. of the 19th EPIA Conf. on Artificial Intelligence. LNCS, vol. 11804, pp. 584–595. Springer (2019)
42. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Advances in Neural Information Processing Systems. vol. 28, pp. 2692–2700. Curran Associates, Inc. (2015)
43. Wickert, T.I., Smet, P., Berghe, G.V.: The nurse rerostering problem: Strategies for reconstructing disrupted schedules. Computers & Operations Research **104**, 319–337 (2019)