Learning to Control Local Search for Combinatorial Optimization

Jonas K. Falkner (✉), Daniela Thyssens, Ahmad Bdeir, and Lars Schmidt-Thieme

Institute for Computer Science, University of Hildesheim, Hildesheim, Germany
{falkner,thyssens,bdeir,schmidt-thieme}@ismll.uni-hildesheim.de

Abstract. Combinatorial optimization problems are encountered in many practical contexts such as logistics and production, but exact solutions are particularly difficult to find and usually NP-hard for considerable problem sizes. To compute approximate solutions, a zoo of generic as well as problem-specific variants of local search is commonly used. However, which variant to apply to which particular problem is difficult to decide even for experts.
In this paper we identify three independent algorithmic aspects of such local search algorithms and formalize their sequential selection over an optimization process as Markov Decision Process (MDP). We design a deep graph neural network as policy model for this MDP, yielding a learned controller for local search called NeuroLS. Ample experimental evidence shows that NeuroLS is able to outperform both, well-known general purpose local search controllers from the field of Operations Research as well as latest machine learning-based approaches.

Keywords: combinatorial optimization · local search · neural networks.

1 Introduction

Combinatorial optimization problems (COPs) arise in many research areas and applications. They appear in many forms and variants like vehicle routing[43], scheduling[41] and constraint satisfaction[44] problems but share some general properties. One of these properties is that most COPs are proven to be NP-hard which makes their solution very complex and time consuming. Over the years many different solution approaches were proposed. *Exact methods* like *branch-and-bound*[26] attempt to find the global optimum of a COP based on smart and efficient ways of searching the solution space. While they are often able to find the optimal solution to small scale COPs in reasonable time, they require a significant amount of time to tackle larger instances of sizes relevant for practical applications. For that reason, *heuristic methods* were proposed which usually cannot guarantee to find the global optimum but sometimes can define a lower bound on the performance, which can be achieved at a minimum, and have shown good empirical performance. One common and well-known heuristic method is *local search* (LS)[1]. The main concept of LS is to iteratively explore the search

space of candidate solutions in the close neighborhood of the current solution by applying small (local) changes. Simple LS procedures, like hill climbing, easily get stuck in bad local optima from which it cannot escape anymore. Therefore, LS is commonly used in combination with *meta-heuristics* which enable the procedure to escape from local optima and achieve better final performance. The meta-heuristics introduced in section 2.2 are well established in the optimization community and have demonstrated very good performance on a plenitude of different COPs.

Since a few years however, there is an increasing interest in leveraging methods from the field of machine learning (ML) to solve COPs, as recent developments in neural network architectures, involving Transformers[47] and Graph Neural Networks (GNNs)[52], have led to significant progress in this domain. Most work based on ML is concerned with auto-regressive approaches to construct feasible solutions[17,24,49,53], but there is also some work which focuses on iterative improvement[6,28,51] or exact solutions[10,33] for COPs. While the existing improvement approaches share some similarities with meta-heuristics and local search, the exact formulation of the methods is often problem specific and misses the generality of the LS framework as well as a clear definition of the intervention points which can be used to control the meta-heuristic procedure.

In this work we present a consistent formulation of learned meta-heuristics in local search procedures and show on two representative problems how it can be successfully applied. In our experiments we compare our method to well-known meta-heuristics commonly used with LS at the example of capacitated vehicle routing (CVRP) and job shop scheduling (JSSP). The results show that our GNN-based learned meta-heuristic consistently outperforms these methods in terms of speed and final performance. In further experiments we also establish our method in the context of existing ML solution approaches.

Contributions

1. We identify and describe three independent algorithmic aspects of local search for COPs, each with several alternatives, and formalize their sequential selection during an iterative search run as Markov Decision Process.
2. We design a deep graph neural network as policy model for this MDP, yielding a learned controller for local search called *NeuroLS*.
3. We provide ample experimental evidence that NeuroLS outperforms both, well-known general purpose local search controllers from the Operations Research literature (so called meta-heuristics) as well as earlier machine learning-based approaches.

2 Related Work

There is a plenitude of approaches and algorithms to tackle combinatorial optimization problems. One common heuristic method is *Local Search* (LS)[1]. In the classical discrete optimization literature it is often embedded into a meta-heuristic procedure to escape local optima.

2.1 Construction Heuristics

Construction algorithms are concerned with finding a first feasible solution for a given COP. They usually start with an empty solution and consecutively assign values to the respective decision variables to construct a full solution. Well-known methods are e.g. the *Savings* heuristic[7] for vehicle routing problems or *priority dispatching rules* (PDR)[5] for scheduling. Most improvement and meta-heuristic methods require a feasible initial solution from which they can start to improve.

2.2 Meta-Heuristics

Meta-heuristics are the go-to method for complex discrete optimization problems. They effectively balance the exploration of the solution space and the exploitation of promising solutions. There are two major types of methods:

Trajectory-based Methods Trajectory-based methods include many well-known approaches which are used in combination with LS. During the search they only maintain a single solution at a time which is iteratively changed and adapted. *Simulated Annealing* (SA) is a probabilistic acceptance strategy based on the notion of controlled cooling of materials first proposed in [22]. The idea is to also accept solutions which are worse than the best solution found so far to enable exploration of the search space but with a decreasing probability to increasingly focus on exploitation the further the search advances. *Iterated Local Search* (ILS)[27] alternates between a diversification and an intensification phase. In the diversification step the current solution is perturbed while the alternating intensification step executes a greedy local search with a particular neighborhood. *Variable Neighborhood Search* (VNS)[31] employs a similar diversification step but changes the type of the applied LS move after each perturbation (in a predefined order) to systematically control the LS neighborhood in the intensification phase. *Tabu Search* (TS) was proposed by Glover[12] and is based on the possible acceptance of non-improving moves and a so called tabu list, a kind of filter preventing moves which would keep the search stuck in local optima. This list acts as a memory which in the simplest case stores the solutions of the last k iterations and prevents changes which would move the current solution back to solutions encountered in recent steps. Instead of using a tabu list, *Guided Local Search* (GLS)[50] relies on penalties for different moves to guide the search. These penalties are often based on problem specific features in the solution, e.g. the edges between nodes in a routing problem, and are added to the original objective function of the problem when the LS gets stuck.

Population-based Methods In comparison to trajectory-based approaches population-based methods maintain a whole pool of different solutions throughout the search. Methods include different evolutionary algorithms, particle swarm optimization and other bio-inspired algorithms such as ant colony optimization [11]. Their main idea is based on different adaption, selection and recombination schemes to refine the solution pool during search in order to find better solutions. While a learned population based meta-heuristic is interesting and potentially promising, in this paper we focus on the impact

and effectiveness of a learned trajectory-based approach. More information on advanced meta-heuristics can be found in [11].

2.3 Machine Learning based Methods

In recent years an increasing number of ML-based methods has been proposed. While some work relies on supervised[19,42,49] or unsupervised[20] learning, most current state-of-the-art methods use reinforcement learning (RL). A large fraction of the work focuses on auto-regressive models which learn to sequentially construct feasible solutions from scratch. Such methods have been proposed for many common COPs including the TSP[3,21], CVRP[24,25,34], CVRP-TW[9] and JSSP[15,36,37,53]. The second type of methods is concerned with improvement approaches. Hudson, Malencia and Prorok[18] propose an LS approach guided by an underlying GNN for the TSP. Chen and Tian[6] design a model to rewrite sub-sequences of the problem solution for the CVRP and JSSP based on a component which selects a specific element of the solution and a second component parameterizing a heuristic move to change that part of the solution. However, their model is limited to specific problem settings with a fixed number of jobs and machines or customers while our approach works seamlessly for different problem sizes, as we show in the experiments in section 5.3. In contrast, the authors in [28] learn a policy that selects a specific LS move at each iteration. However, their method incurs prohibitively large computation times and for this reason is not competitive with any of the recent related work [23,25,29]. The authors in [16] learn a repair operator to re-construct heuristically destroyed solutions in a Large Neighborhood Search. Finally, da Costa et al.[35] learn a model to select node pairs for 2-opt moves in the TSP while Wu et al.[51] learn a similar pair-wise selection scheme for 2-opt, node swap and relocation moves in TSP and CVRP. The authors in [29] further improve on the method in [51] by introducing the Dual-aspect collaborative Transformer (DACT) model. Although there exist advanced inference approaches[15] to further improve the performance of auto-regressive ML methods on COPs, here we focus on the vanilla inference via greedy decoding or sampling.

While these methods share some similarities with our approach, they ignore the importance of being able to reject unpromising moves to escape local optima, whereas our approach specifically focuses on this important decision point to effectively control the search. Moreover, our approach is also able to learn when to apply a particular perturbation if the rejection of a sequence of moves is not sufficient for exploration. A detailed overview of the current machine learning approaches to COPs is given in [4,30].

3 Preliminaries

3.1 Problem Formulation

A Combinatorial Optimization Problem Ω is defined on its domain D_Ω which is the set of its instances $x \in D_\Omega$. A COP instance x is usually given by a pair

(S_Ω, f_Ω) of the solution space S consisting of all feasible solutions to Ω and a corresponding cost function $f : S \to \mathbb{R}$. Combinatorial optimization problems normally are either to be minimized or maximized. In this paper we consider all COPs to be problems for which the cost of a corresponding objective function has to be minimized. The main concern is to find a solution $s^* \in S$ representing a *global optimum*, i.e. $f(s^*) \leq f(s)\ \forall s \in S$.

3.2 Local Search

LS is a heuristic search method which is based on the concept of neighborhoods. A neighborhood $\mathcal{N}(s) \subseteq S$ of solution $s \in S$ represents a set of solutions which are somehow close to s. This "closeness" is defined by the neighborhood function $\mathcal{N}^\varphi$ w.r.t. some problem specific operator $\varphi \in \Phi_\Omega$ (e.g. all solutions which can be reached from the current solution by an exchange of nodes). Moreover, we always consider s to be part of its own neighborhood, i.e. $s \in \mathcal{N}(s)$. Then the *local optimum* $\hat{s}$ in the neighborhood $\mathcal{N}$ satisfies $f(\hat{s}) \leq f(s)\ \forall s \in \mathcal{N}(\hat{s})$. A general LS procedure (see algorithm 1) iterates through the neighborhood $\mathcal{N}(s)$ of the current solution s until it finds the local optimum $\hat{s}$.

Algorithm 1: Local Search

input: cost function f, solution s, neighborhood function $\mathcal{N}$, acceptance rule `accept`, stopping rule `stop`,

```
while not stop(s) do
    find s' ∈ N(s) for which accept(s, s')
    s ← s'
return s
```

3.3 Meta-Heuristics

Meta-heuristics wrap an LS procedure to enable it to escape from local optima and to explore the solution space more efficiently. Some of the most common meta-heuristic strategies were described in section 2.2. Algorithm 2 describes a general formulation of a trajectory-based meta-heuristic procedure. Each particular strategy takes different decisions about restarting or perturbing the current solution, configuring the local search and accepting intermediate candidate solutions s' (see algorithm 1). Some decisions can be fixed and are treated as hyper-parameters for some methods. For example, an SA procedure normally does not select a specific neighborhood but just decides about acceptance during the LS. Other approaches like VNS greedily accept all improving moves but apply a perturbation and select a new operator neighborhood every time a local optimum has been reached.

Algorithm 2: Meta-Heuristic (trajectory-based)

input: Solution space S, cost function f, stopping criterion

```
1 s ← construct(S)                          // Construct initial solution
2 while not stopping criterion do
3 |   s ← perturb(S, s)                    // Decide if to perturb/restart
4 |   N ← GetNeighborhood(S, s)            // Define search neighborhood
5 |   s ← LocalSearch(f, s, N, accept, stop) // Execute local search
6 return s
```

4 Proposed Method

4.1 Intervention Points of Meta-Heuristics for Local Search

The application of meta-heuristic strategies to an underlying LS involves several points of intervention, at which decisions can be made to help the search escape local optima in order to guide it towards good solutions. In the following, we define three such intervention points that have a significant impact on the search:

1. *Acceptance*: The first intervention point is the acceptance of candidate solutions s' in an LS step (algorithm 1, line 2). A simple hill climbing heuristic is completely greedy and only accepts improving moves, which often leaves the search stuck in local optima very quickly. In contrast, other approaches like SA will also accept non-improving moves with some probability.
2. *Neighborhood*: The second possible decision a meta-heuristic can make is the selection of a particular operator φ that defines the neighborhood function $\mathcal{N}^{\varphi}$ for the LS (algorithm 2, line 4). Possible operators for scheduling or routing problems could for example be a node exchange. While many standard approaches like SA and ILS only use one particular neighborhood which they treat as a hyper-parameter, VNS is an example for a method that selects a different operator defining a particular neighborhood at each step.
3. *Perturbation*: Many meta-heuristics like ILS and VNS employ perturbations $\psi \in \Psi_{\Omega}$ to the current solution to move to different regions of the search space and escape particularly persistent local optima. Such a perturbation can simply be a restart from a new stochastically constructed initial solution, a random permutation of (a part of) the current solution or a random sequence of operations. The decision *when* to employ a perturbation (algorithm 2, line 5) is commonly done w.r.t. a specific pre-defined number of steps without improvement.

4.2 Meta-Heuristics as Markov Decision Process

In this section we formulate meta-heuristics in terms of an MDP[40] to enable the use of RL approaches to learn a parameterized policy to replace them. In general an MDP is given by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}(s_t, a_t), \mathcal{R}(s_t, a_t))$ representing the

set of states $\mathcal{S}$, the set of actions $\mathcal{A}$, transition probability function $\mathcal{P}(s_t, a_t)$ and reward function $\mathcal{R}(s_t, a_t)$. For our method we define these entities in terms of meta-heuristic decisions as follows:

States We define the state s_t of the problem at time step t with slight abuse of notation as the solution s at time step t, combined with 1) its cost $f(s)$, 2) the cost $f(\hat{s}_t)$ of the best solution found so far, 3) the last acceptance decision, 4) the last operator used, 5) current time step t, 6) number of LS steps without improvement and 7) the number of perturbations or restarts.

Actions Depending on the policy we want to train, we define the action set as the combinatorial space of

1. *Acceptance* decisions: a boolean decision variable of either accepting or rejecting the last LS step
$$\mathcal{A}_{\text{A}} \coloneqq \{0, 1\}, \tag{1}$$
2. *Acceptance-Neighborhood* decisions: the joint space of the acceptance of the last move and the set of possible operators $\varphi \in \Phi$ which define the search neighborhood(s) $\mathcal{N}^{\varphi}$ for the next step
$$\mathcal{A}_{\text{AN}} \coloneqq \{0, 1\} \times \Phi, \tag{2}$$
3. *Acceptance-Neighborhood-Perturbation* decisions: the joint space of acceptance and the combined sets of operators $\varphi \in \Phi$ and perturbations $\psi \in \Psi$
$$\mathcal{A}_{\text{ANP}} \coloneqq \{0, 1\} \times \{\Phi \cup \Psi\}. \tag{3}$$

Transitions The transition probability function $\mathcal{P}(s_t, a_t)$ models the state transition from state s_t to the next state s_{t+1} depending on action a_t representing the acceptance decision, the next operator φ and a possible perturbation or restart, which is part of the problem state (or action in case of $\mathcal{A}_{\text{ANP}}$).

Rewards The reward function $\mathcal{R}(s_t, a_t)$ gives the reward for a transition from state s_t to the next state s_{t+1}. Here we define the reward r_t as the relative improvement of the last LS step (defined by action a_t) w.r.t. the cost of the best solution found until t and clamped at 0 to avoid negative rewards:
$$r_t \coloneqq \max(f(\hat{s}_t) - f(s_{t+1}), 0) \tag{4}$$

Policy We employ Deep Q-Learning[46] and parameterize the learned policy π_θ via a softmax over the corresponding Q-function $Q_\theta(s_t, a_t)$ which is represented in turn by our GNN-based encoder-decoder model with trainable parameters θ:
$$\pi_\theta(a_t \mid s_t) = \frac{\exp(Q_\theta(s_t, a_t))}{\sum_{\mathcal{A}} \exp(Q_\theta(s_t, a_t))}. \tag{5}$$

4.3 Model Architecture

In this section we describe our encoder and decoder models to parameterize $Q_\theta(s_t, a_t)$. Many COPs like routing and scheduling problems have an underlying

graph structure which can be used when encoding these problems, providing an effective inductive bias for the corresponding learning methods. In general we assume a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the set of nodes $\mathcal{V}$, $N = |\mathcal{V}|$ and the set of directed edges $\mathcal{E} \subseteq \{(i,j) \subseteq \mathcal{V}\}$. Moreover, we assume an original node feature matrix $X \in \mathbb{R}^{N \times d_{\text{in}}}$ and edge features $e_{i,j} \in \mathbb{R}$ for each edge (i,j). To leverage this structural information many authors have used Recurrent Neural Networks [3,6,49], Transformers[24,25,29,47] or Graph Neural Networks (GNN)[19,35,53].

Encoder

Since edge weights are very important in COP graphs, we employ a simple version of the GNN operator proposed in [32] with GELU[14] activations which can directly work with edge weights and outperformed GAT[48] and ReLU in preliminary experiments. The resulting GNN layer is defined as:

$$\begin{aligned} h_i^{(l)} &= \text{GNN}^{(l)}(h_i^{(l-1)}) \\ &= \text{GELU}\left(\text{MLP}_1^{(l)}(h_i^{(l-1)}) + \text{MLP}_2^{(l)}(\sum_{j \in \mathcal{H}(i)} e_{j,i} \cdot h_j^{(l-1)})\right), \end{aligned} \tag{6}$$

where $h_i^{(l-1)} \in \mathbb{R}^{1 \times d_{\text{emb}}}$ is the latent feature embedding of node i at the previous layer $l-1$, $\mathcal{H}(i)$ is the 1-hop graph neighborhood of node i, $\text{MLP}_1^{(l)}$ and $\text{MLP}_2^{(l)}$ are Multi-Layer Perceptrons $\text{MLP} : \mathbb{R}^{d_{\text{emb}}} \to \mathbb{R}^{d_{\text{emb}}}$. Furthermore, we add residual connections and layer normalization[2] to each layer.

In the first layer the latent feature vector $h_i^{(0)}$ is created by feeding the original node features x_i into an $\text{MLP} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{emb}}}$:

$$h_i^{(0)} = \text{MLP}^{(0)}(x_i). \tag{7}$$

and another $\text{MLP}^{(L)} : \mathbb{R}^{d_{\text{emb}}} \to \mathbb{R}^{d_{\text{emb}}}$ is placed at the end of the GNN stack.

In order to further leverage structural information, we introduce 3 stages to compute the latent embeddings. The first stage uses the edge set $\mathcal{E}^{\text{stat}}$ of the *static* problem graph. For the CVRP we use the graph induced by the K nearest neighbors of each node, for scheduling problems the Directed Acyclic Graph (DAG) representing the predefined order of operations for each job. This edge set is fixed and does not change throughout the search. In contrast, the second stage utilizes the edge set $\mathcal{E}^{\text{dyna}}$ representing the *dynamic* problem graph which usually changes in every LS step, e.g. the edges constituting the different tours in routing problems or the machine graph which represents the sequence of jobs on each machine for scheduling. Our proposed network architecture consists of L^{stat} GNN layers for the static graph, followed by L^{dyna} layers which propagate over the dynamic graph. Finally, we add another layer, again using the static edges, to consolidate the dynamic information over the static graph, leading to a total of $L = L^{\text{stat}} + L^{\text{dyna}} + 1$ GNN layers.

The final stage serves to refine the embedding via aggregation based on the dynamic information of group membership which is present in the solution. Each node normally belongs to one of K (not necessarily disjoint) groups $\mathcal{M}_k$ of the solution, e.g. to a particular tour or machine. Following this idea we pool the

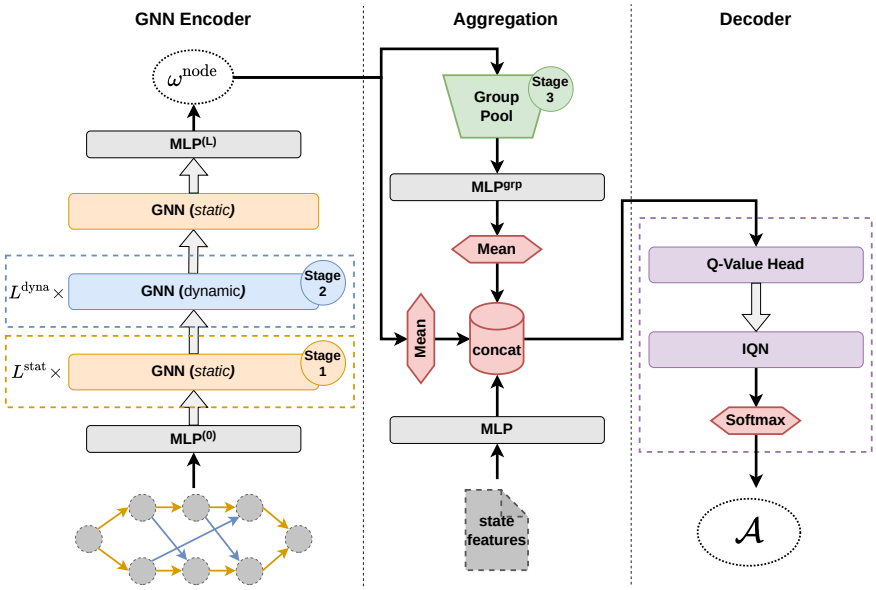

Fig. 1. Visualization of the NeuroLS model architecture.

final embeddings ω_i^{node} from the GNN stack w.r.t. their membership and feed them through another MLP:

$$\omega_k^{\text{grp}} = \text{MLP}^{\text{grp}}\left([\text{MAX}(\omega_i^{\text{node}} \mid i \in \mathcal{M}_k); \text{MEAN}(\omega_i^{\text{node}} \mid i \in \mathcal{M}_k)]\right), \quad (8)$$

with max and mean pooling $\mathbb{R}^{N\times d_{\text{emb}}} \rightarrow \mathbb{R}^{K\times d_{\text{emb}}}, K << N$ over the node dimension N, $\mathcal{M}_k$ as membership to the k-th group (k-th tour, k-th machine, etc.) and $[\,;]$ representing concatenation in the embedding dimension d_{emb}.

Finally, the additional features of the state representation (current cost, best cost, last acceptance, etc.) described in the last section, are concatenated and projected by a simple linear layer to create an additional latent feature vector $\omega^{\text{feat}} \in \mathbb{R}^{d_{\text{emb}}}$. We provide a runtime and memory analysis in Appendix B.

Decoder

Our decoder takes the different embeddings created in the encoder, aggregates the node embeddings $\omega^{\text{node}} \in \mathbb{R}^{N\times d_{\text{emb}}}$ and group embeddings $\omega^{\text{grp}} \in \mathbb{R}^{K\times d_{\text{emb}}}$ via a simple mean over the node and group dimension and concatenates them with the feature embedding $\omega^{\text{feat}} \in \mathbb{R}^{d_{\text{emb}}}$. This representation is the input to a final 2-layer MLP regression head $\mathbb{R}^{3*d_{\text{emb}}} \rightarrow \mathbb{R}^{|\mathcal{A}|}$ which outputs the value predictions of the Q-function. The full architecture is shown in figure 1.

4.4 Reinforcement Learning Algorithm

To train our policy model we employ Double Deep Q-Learning[46] with n-step returns[40] and Implicit Quantile Networks (IQN)[8]. IQNs enable a distributional formulation of Q-Learning where the Deep Q-Network is trained w.r.t. an

underlying value distribution represented by a learned quantile function instead of single point estimates. In order to represent this learned quantile function an IQN is introduced as a small additional neural network which is jointly trained to transform samples from a base distribution (e.g. uniform) to the respective quantile values of the target distribution, i.e. the distribution over the returns.

5 Experiments

5.1 Applications

Job Shop Scheduling Problem (JSSP) The JSSP is concerned with scheduling a number of jobs J on a set of K machines denoted by M. Each job consists of a sequence of operations O_{ij} with fixed processing times p_{ij} which need to be processed in a predefined order. In the simplest problem variant every job has exactly one operation on each machine. A solution to the problem consists of the exact order of the operations on all machines. In this paper we choose to minimize the *makespan*, which is the longest time span from start of the first operation until the end of the last one to finish, corresponding to the longest path in the respective DAG representation of the problem. We denote the size of a JSSP instances as $|J| \times |M|$ and follow [53] in creating instances for training and validation by sampling processing times from a uniform distribution.

Capacitated Vehicle Routing Problem (CVRP) The CVRP consists of a set of N customer nodes and a depot node. It is concerned with serving the demands q_i of the customer nodes in tours starting and ending at the depot node by employing K homogeneous vehicles with a fixed capacity $Q > 0$. The tour of vehicle $k \in K$ is a sequence of indices w.r.t. a subset of all customers nodes representing the order in which vehicle k visits the respective nodes. A set of feasible tours serving all customer nodes represents a solution to the problem, whereas the objective is to minimize the total length of all tours. We follow [24] in creating the training and validation sets by generating instances with coordinates uniformly sampled in the unit square.

5.2 Setup

For the JSSP we implement a custom LS solver in python. It implements four different node moves and a perturbation operator based on a sequence of such moves. As construction heuristic and for restarts we implement several stochastic variants of common PDRs (see appendix D for more details). The LS for the CVRP uses the C++ based open source solver VRPH[13] for which we implement a custom wrapper and interface to expose and support the necessary intervention points. VRPH includes several different LS moves and perturbation operators including 2-opt, 3-opt and different exchange and point moves.

Preliminary experiments showed that the CET and 2-opt moves performed best for the JSSP and CVRP respectively, when used for meta-heuristics which

do not select the operator. Thus, we employ these operators in our main experiments. We train all our models for 80 epochs with 19200 transitions each and pick the model checkpoint with the best validation performance.

Hyperparameters for NeuroLS and all meta-heuristics are tuned on a validation set consisting of 512 generated random instances. This is in contrast to most classical approaches which fine tune their hyper-parameters directly on the benchmark dataset. We argue that our approach facilitates a better and more objective comparison between these methods. Further details on hyper-parameters can be found in Appendix C and we provide our code on github[1].

We train 3 different types of policies for NeuroLS, one for each of the action spaces described in section 4.2. In the experiments we denote these policies as NLS_A, NLS_AN and NLS_ANP. As analyzed in [51], random solutions do not provide a good starting point for improvement approaches. For that reason we initialize the solutions of NeuroLS and the meta-heuristics with the FDD/MWKR PDR[39] for scheduling and the savings construction heuristic[7] for the CVRP.

5.3 Results

Table 1. Results of state-of-the-art machine learning based construction methods and local search approaches (100 iterations) on the Taillard benchmark[41]. For instances of size 50x15, 50x20 and 100x20 we use the NeuroLS model trained on instances of size 30x15 and 30x20 respectively. Percentages are the average gap to the best known upper bound. Best gap is marked in **bold**.

Model	**Instance size**								**Avg**
	15x15	20x15	20x20	30x15	30x20	50x15	50x20	100x20	
ML-based									
L2D[53]	25.92%	30.03%	31.58%	32.88%	33.64%	22.35%	26.37%	13.64%	27.05%
L2S[37]	20.12%	24.83%	29.25%	24.59%	31.91%	15.89%	21.39%	9.26%	22.16%
SN[36]	15.32%	19.43%	17.23%	18.95%	23.75%	13.83%	13.56%	6.67%	16.09%
Meta-heuristic + Local Search									
SA	13.92%	17.01%	17.16%	17.53%	21.59%	12.50%	13.11%	6.61%	14.93%
SA_restart	13.77%	17.01%	17.57%	17.62%	21.78%	12.54%	13.22%	6.75%	15.03%
ILS	11.57%	13.57%	13.85%	16.07%	18.72%	12.65%	12.15%	6.72%	13.16%
ILS+SA	13.32%	16.05%	15.38%	16.93%	19.74%	13.07%	13.43%	7.08%	14.37%
VNS	9.96%	13.71%	14.51%	15.77%	18.69%	11.64%	11.92%	6.26%	12.81%
NeuroLS									
NLS_A	**9.76**%	13.33%	13.02%	15.29%	17.94%	11.81%	11.96%	6.33%	12.43%
NLS_AN	10.32%	**13.18**%	**12.95**%	**14.91**%	17.78%	11.87%	12.02%	6.22%	**12.41**%
NLS_ANP	10.49%	16.32%	15.24%	15.35%	**17.64**%	**11.62**%	**11.76**%	**6.09**%	13.06%

[1] https://github.com/jokofa/NeuroLS

Table 2. Results of state-of-the-art machine learning based methods and local search approaches (200 iterations) on the Uchoa benchmark[45]. For all instances sizes we use the NeuroLS model trained on instances of size 100. Percentages are the average gap to the best known solution. Best gap is marked in **bold**.

Model	Instance Group								Avg
	n100		*n150*		*n200*		*n250*		
	cost	time	cost	time	cost	time	cost	time	
ML-based									
POMO[25]	17.24%	0.3	12.81%	0.4	20.52%	0.4	15.14%	0.4	16.43%
POMO[25] (aug)	6.32%	**0.1**	9.41%	**0.1**	13.47%	**0.1**	9.21%	**0.1**	9.60%
DACT[29]	13.19%	15.1	20.26%	21.2	16.91%	27.2	24.93%	33	18.82%
DACT[29] (aug)	11.52%	16.6	17.75%	23.1	15.34%	29.8	21.86%	37.8	16.62%
Meta-heuristic + Local Search									
ORT[38] GLS	6.91%	9.4	10.46%	10.9	**7.80**%	13.4	12.12%	5.0	9.32%
ORT[38] TS	6.78%	39.0	10.55%	109.5	7.85%	126.6	12.13%	7.0	9.33%
SA	6.92%	0.7	5.79%	1.3	16.63%	2.1	5.92%	3.3	8.81%
$SA_{restart}$	6.96%	0.7	5.79%	1.3	16.67%	2.0	5.99%	3.0	8.85%
ILS	6.56%	0.8	5.96%	1.5	16.73%	2.4	6.08%	3.7	8.83%
ILS+SA	6.94%	0.6	6.01%	1.2	16.72%	1.9	6.04%	2.8	8.93%
VNS	7.99%	0.4	6.55%	0.7	17.27%	0.9	6.36%	1.4	9.54%
NeuroLS									
NLS_A	5.43%	1.5	5.23%	2.4	15.97%	3.6	5.22%	5.3	7.96%
NLS_{AN}	**5.42**%	1.7	**4.90**%	2.7	15.85%	4.0	**5.08**%	5.9	**7.81**%
NLS_{ANP}	**5.42**%	1.7	**4.90**%	2.7	15.85%	4.0	**5.08**%	6.1	**7.81**%

JSSP We evaluate all methods on the well-known benchmark dataset of Taillard[41]. It consists of 80 instances of size 15x15 up to 100x20. We compare our model against common meta-heuristic baselines including SA, SA with restarts, ILS, ILS with SA acceptance and VNS. Moreover, we report the results of three recent state-of-the-art ML-based approaches: *Learning to dispatch* (L2D)[53], *Learning to schedule* (L2S)[37] and *ScheduleNet* (SN)[36]. We follow [53] in training different models for problem sizes 15x15 up to 30x20 and apply the 30x15 and 30x20 model to larger instances of size 50x15, 50x20 and 100x20 to evaluate its generalization capacity. All models are trained for 100 LS iterations but evaluated for 50-200. The aggregated results per group of same size are shown in table 1 (for per instance results see appendix F). Results are reported as percentage gaps to the best known solution.

First of all, the results show that NeuroLS is able to outperform all other meta-heuristics on all sizes of instances. The different policies differ in how well they work for different problem sizes. While NLS_A works best for the smallest 15x15 instances, it is outperformed by NLS_{AN} on medium sized instances and NLS_{ANP} achieves the best results for large instances. This is to some extent expected, since the effect of specific LS operators and more precise perturbations is greater for larger instances while this does not seem to be necessary for rather

small instances. VNS is the best of the meta-heuristic approaches which is able to beat NLS_{ANP} on the smaller instances and NLS_{A} and NLS_{AN} at least on some of the larger ones. In general, the iterative methods based on LS achieve much better results than the ML-based auto-regressive methods, which can be seen by the large improvements that can be achieved in just 100 iterations, reducing the gap by an average of 3.7% compared to the best ML-method SN[36].

CVRP For the CVRP we use the recent benchmark dataset of Uchoa et al.[45] and select all instance up to 300 customer nodes. We define four groups of instances with 100-149 nodes as *n100*, 150-199 nodes as *n150*, 200-249 as *n200* and 250-299 as *n250*. We compare against the same meta-heuristics mentioned above and additionally to GLS and TS provided by the OR-Tools (ORT) library[38]. Furthermore, we compare to the recent state-of-the-art ML approaches POMO[25] and DACT[29] which outperformed all other ML methods mentioned in section 2.3 in their experiments and provide open source code, which is why we consider them to be sufficient for a suitable comparison.

Since most ML-based methods (POMO, DACT, etc.) do not respect the maximum vehicle constraint for all instances of the Uchoa benchmark, we follow [29] in removing this constraint and treat the benchmark dataset as a highly diverse test set w.r.t. the distributions of customers and number of required vehicles. This is also consistent with the general goal of the ML-based methods, which is not to achieve the best known results but to find sufficiently good results in reasonable time. In this case we control the computational resources spent on the search by specifying a particular number of iterations for the search. Furthermore, we evaluate all models with a batch size of 1.

The results presented in table 2 show that NeuroLS is able to outperform the considered meta-heuristic approaches on all instance groups. Moreover, our approach also outperforms the state-of-the-art ML-based methods on all groups but *n200*, where POMO and DACT with additional instance augmentations (aug) outperform NeuroLS by a small margin. The OR-Tools implementation of GLS and TS outperforms our method only on the *n200* instances, although they require prohibitively large runtimes (wall-clock time). In terms of runtimes we also outperform DACT by a magnitude, while the learned auto-regressive construction method POMO is the fastest overall. Finally, the experiment results also show that our method is able to generalize to problem sizes as well as numbers of iterations unseen in training. We show this on the JSSP instances of size 50x15, 50x20 and 100x20 and for all Uchoa instances for the CVRP, which are all larger than the 100 node instances used during training.

6 Conclusion

In this paper we identify three important intervention points in meta-heuristics for local search on COPs and incorporate them in a MDP. We then design a GNN-based controller which is trained with RL to parameterize three types of learned meta-heuristics. The resulting methods learn to control the LS by deciding about acceptance, neighborhood selection and perturbations. In com-

prehensive experiments on two common COPs in scheduling and vehicle routing, NeuroLS outperforms several well-known meta-heuristics as well as state-of-the-art ML-based approaches, confirming the efficacy of our method.

For future work we consider more fine-grained interventions, e.g. to restrict the search neighborhood and to replace the problem graph with a graph representation of the corresponding LS graph, in which every node represents a feasible solution together with its respective cost.

Acknowledgements This work was supported by the German Federal Ministry of Education and Research (BMBF), project "Learning to Optimize" (01IS20013A:L2O) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK), within the IIP-Ecosphere project (01MK20006D).

References

1. Aarts, E., Aarts, E.H., Lenstra, J.K.: Local search in combinatorial optimization. Princeton University Press (2003)
2. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
3. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940 (2016)
4. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research **290**(2), 405–421 (2021)
5. Blackstone, J.H., Phillips, D.T., Hogg, G.L.: A state-of-the-art survey of dispatching rules for manufacturing job shop operations. The International Journal of Production Research **20**(1), 27–45 (1982)
6. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization. Advances in Neural Information Processing Systems **32** (2019)
7. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations research **12**(4), 568–581 (1964)
8. Dabney, W., Ostrovski, G., Silver, D., Munos, R.: Implicit quantile networks for distributional reinforcement learning. In: International conference on machine learning. pp. 1096–1105. PMLR (2018)
9. Falkner, J.K., Schmidt-Thieme, L.: Learning to solve vehicle routing problems with time windows through joint attention. arXiv preprint arXiv:2006.09100 (2020)
10. Gasse, M., Chételat, D., Ferroni, N., Charlin, L., Lodi, A.: Exact combinatorial optimization with graph convolutional neural networks. Advances in Neural Information Processing Systems **32** (2019)
11. Gendreau, M., Potvin, J.Y., et al.: Handbook of metaheuristics, vol. 2. Springer (2010)
12. Glover, F.: Future paths for integer programming and links to artificial intelligence. Computers & operations research **13**(5), 533–549 (1986)
13. Groër, C., Golden, B., Wasil, E.: A library of local search heuristics for the vehicle routing problem. Mathematical Programming Computation **2**(2), 79–101 (2010)
14. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)

15. Hottung, A., Kwon, Y.D., Tierney, K.: Efficient active search for combinatorial optimization problems. arXiv preprint arXiv:2106.05126 (2021)
16. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. arXiv preprint arXiv:1911.09539 (2019)
17. Hu, H., Zhang, X., Yan, X., Wang, L., Xu, Y.: Solving a new 3d bin packing problem with deep reinforcement learning method. arXiv preprint arXiv:1708.05930 (2017)
18. Hudson, B., Li, Q., Malencia, M., Prorok, A.: Graph neural network guided local search for the traveling salesperson problem. arXiv preprint arXiv:2110.05291 (2021)
19. Joshi, C.K., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the travelling salesman problem. arXiv preprint arXiv:1906.01227 (2019)
20. Karalias, N., Loukas, A.: Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. Advances in Neural Information Processing Systems **33**, 6659–6672 (2020)
21. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. Advances in neural information processing systems **30** (2017)
22. Kirkpatrick, S., Gelatt Jr, C.D., Vecchi, M.P.: Optimization by simulated annealing. science **220**(4598), 671–680 (1983)
23. Kool, W., van Hoof, H., Gromicho, J., Welling, M.: Deep policy dynamic programming for vehicle routing problems. arXiv preprint arXiv:2102.11756 (2021)
24. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! arXiv preprint arXiv:1803.08475 (2018)
25. Kwon, Y.D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: Pomo: Policy optimization with multiple optima for reinforcement learning. Advances in Neural Information Processing Systems **33**, 21188–21198 (2020)
26. Lawler, E.L., Wood, D.E.: Branch-and-bound methods: A survey. Operations research **14**(4), 699–719 (1966)
27. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: Framework and applications. In: Handbook of metaheuristics, pp. 129–168. Springer (2019)
28. Lu, H., Zhang, X., Yang, S.: A learning-based iterative method for solving vehicle routing problems. In: International conference on learning representations (2019)
29. Ma, Y., Li, J., Cao, Z., Song, W., Zhang, L., Chen, Z., Tang, J.: Learning to iteratively solve routing problems with dual-aspect collaborative transformer. Advances in Neural Information Processing Systems **34** (2021)
30. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research **134**, 105400 (2021)
31. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & operations research **24**(11), 1097–1100 (1997)
32. Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M.: Weisfeiler and leman go neural: Higher-order graph neural networks. In: Proceedings of the AAAI conference on artificial intelligence. vol. 33, pp. 4602–4609 (2019)
33. 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)
34. Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. Advances in neural information processing systems **31** (2018)

35. d O Costa, P.R., Rhuggenaath, J., Zhang, Y., Akcay, A.: Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In: Asian Conference on Machine Learning. pp. 465–480. PMLR (2020)
36. Park, J., Bakhtiyar, S., Park, J.: Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. arXiv preprint arXiv:2106.03051 (2021)
37. Park, J., Chun, J., Kim, S.H., Kim, Y., Park, J.: Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. International Journal of Production Research **59**(11), 3360–3377 (2021)
38. Perron, L., Furnon, V.: Or-tools, https://developers.google.com/optimization/
39. Sels, V., Gheysen, N., Vanhoucke, M.: A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. International Journal of Production Research **50**(15), 4255–4270 (2012)
40. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
41. Taillard, E.: Benchmarks for basic scheduling problems. european journal of operational research **64**(2), 278–285 (1993)
42. Thyssens, D., Falkner, J., Schmidt-Thieme, L.: Supervised permutation invariant networks for solving the cvrp with bounded fleet size. arXiv preprint arXiv:2201.01529 (2022)
43. Toth, P., Vigo, D.: The vehicle routing problem. SIAM (2002)
44. Tsang, E.: Foundations of constraint satisfaction: the classic text. BoD–Books on Demand (2014)
45. Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., Subramanian, A.: New benchmark instances for the capacitated vehicle routing problem. European Journal of Operational Research **257**(3), 845–858 (2017)
46. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence. vol. 30 (2016)
47. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
48. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
49. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. Advances in neural information processing systems **28** (2015)
50. Voudouris, C., Tsang, E.P., Alsheddy, A.: Guided local search. In: Handbook of metaheuristics, pp. 321–361. Springer (2010)
51. Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A.: Learning improvement heuristics for solving routing problems.. IEEE Transactions on Neural Networks and Learning Systems (2021)
52. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems **32**(1), 4–24 (2020)
53. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Chi, X.: Learning to dispatch for job shop scheduling via deep reinforcement learning. Advances in Neural Information Processing Systems **33**, 1621–1632 (2020)