Graph Reinforcement Learning for Operator Selection in the ALNS Metaheuristic

Graph Reinforcement Learning for Operator Selection in the ALNS Metaheuristic

Syu-Ning Johnn*[1], Victor-Alexandru Darvariu*[2,3], Julia Handl[4], and Joerg Kalcsics[1]

[1] University of Edinburgh
shunee.johnn@sms.ed.ac.uk; joerg.kalcsics@ed.ac.uk
[2] University College London v.darvariu@cs.ucl.ac.uk
[3] The Alan Turing Institute
[4] University of Manchester Julia.Handl@manchester.ac.uk

Abstract. ALNS is a popular metaheuristic with renowned efficiency in solving combinatorial optimisation problems. However, despite 16 years of intensive research into ALNS, whether the embedded adaptive layer can efficiently select operators to improve the incumbent remains an open question. In this work, we formulate the choice of operators as a Markov Decision Process, and propose a practical approach based on Deep Reinforcement Learning and Graph Neural Networks. The results show that our proposed method achieves better performance than the classic ALNS adaptive layer due to the choice of operator being conditioned on the current solution. We also discuss important considerations such as the size of the operator portfolio and the impact of the choice of operator scales. Notably, our approach can also save significant time and labour costs for handcrafting problem-specific operator portfolios.

Keywords: Adaptive Large Neighbourhood Search · Markov Decision Process · Deep Reinforcement Learning · Graph Neural Networks

1 Introduction

Adaptive large neighbourhood search (ALNS) is a metaheuristic introduced by Ropke and Pisinger [18] to solve combinatorial optimisation problems (COPs) that iteratively deconstructs and reconstructs a part of the solution in the search for more promising solutions. This "relax-and-reoptimise" process is executed via a pair of destroy and repair heuristics called operators. Based on the principle of Shaw's large neighbourhood search (LNS) [21], ALNS contains multiple operators and an adaptive layer that iteratively selects and applies different operator pairs from a predefined operator portfolio. This is typically an embedded Roulette Wheel (RW) algorithm that selects operators in a probabilistic fashion.

ALNS is renowned for its efficiency in finding good-quality solutions within reasonable computational time. However, despite the wide use of ALNS for solving various COPs, the ways in which each ALNS component contributes to its general performance is not well understood. A recent ALNS state-of-the-art review [12] indicated that only 2 out of 252 papers go beyond the straightforward

implementation and concentrate on component-based analysis, including [19] which focuses on the selection of the ALNS acceptance criterion, and [25] on the effectiveness of the ALNS adaptive layer for operator selection.

We summarise two main deficiencies that exist in the current ALNS framework. Firstly, studies have shown that the adaptive layer has limited capability to dynamically select the best operators, despite being engineered to do so. Turkeš et al. [25] reported a mere 0.14% average improvement brought by the adaptive layer from the analysis of 25 ALNS implementations, indicating a need for a more efficient operator selection mechanism that reflects the contribution of individual operators accurately. Secondly, operator portfolio design for a particular problem can require considerable manual evaluation [12]. The choice of portfolio size is also delicate: too few operators might not enable the search to visit unexplored neighbourhoods, but a plethora of operators can introduce noise to the adaptive layer. To mitigate these deficiencies, we make the following contributions:

- We formulate the choice of a sequence of operators as a Markov Decision Process (MDP), in which an agent receives a reward proportional to the improvement in the solution. We draw a correspondence between value-based Reinforcement Learning (RL) methods used to solve MDPs, such as Q-learning, and the classic RW update used in ALNS. A key insight is that RL estimates are conditioned on the current solution, while RW updates are independent of it, which indicates the potential to learn a stronger operator selector through the RL framework;
- We propose a practical approach based on Deep RL for learning to select operators. Furthermore, we highlight the potential of Graph Neural Networks (GNNs) for generalizing to larger problem instances than seen during training;
- We carry out an extensive evaluation that includes a large selection of representative operators from the literature. Our results demonstrate that the proposed approach performs significantly better than the RW mechanism. We also analyse the impact of important practical considerations such as portfolio sizes and destroy operator scales on the optimality of the solutions.

2 Literature Review

In the last decade, training Machine Learning (ML) methods to solve highly complex COPs has become increasingly prominent [3], especially for the Vehicle Routing Problem (VRP) and its variants [1]. Several pioneering studies applied RL to directly construct solutions for routing-related problems. Bello et al. [2] used policy gradient algorithms to tackle the Travelling Salesperson Problem (TSP). Nazari et al. [14] proposed an end-to-end framework that outputs solutions directly from the routing-based problem instances. Moreover, Kool et al. [10] proposed a construction heuristic that consists of an attention-based decoder trained with RL to regressively build solutions for the TSP and its variants.

ML can also be applied in many cases to enhance existing solution approaches, especially in the field of metaheuristics [24]. The reader is referred

to the work of Karimi-Mamaghan et al. [8] for a comprehensive review on the integration of ML and metaheuristics to tackle COPs.

Several recent studies focused on integrating ML with classic LNS, which can be viewed as a simplified version of the ALNS metaheuristic without the adaptive layer for operator selection. As the first paper on this topic, Hottung and Tierney [7] proposed 2 generalised random-based destroy operators and a single repair operator with automated learning based on a deep neural network with an attention mechanism. Their work was the first to consider the application of RL to LNS for solving a VRP, and achieved solutions of better quality than classic optimisation approaches. Nevertheless, their proposed learning mechanism only focuses on repairing incomplete solutions during the repair phase. In another work, Falkner et al. [6] integrated a pre-trained neural construction heuristic as the repair operator in the LNS framework to solve the VRP with time windows. The destroy procedures remain handcrafted and are classified into 2 groups without any learning involved. Moreover, Oberweger et al. [15] enhanced the LNS framework with an ML-guided destroy operator to solve a staff rostering problem. For the reconstruction phase, the authors developed a mixed-integer linear program as a repair method. Lastly, Syed et al. [23] proposed a neural network in an LNS setting to solve a vehicle ride-hailing problem. However, it uses supervised learning, which requires a large training dataset and, furthermore, can only perform as well as the algorithm used for its generation.

A very recent concurrent work by Reijnen et al. [17] also applies Deep RL to improve ALNS operator selection. It considers a state space that only uses information about the search status (such as the search step), ignoring information about the current solution. In contrast to this, the design of our approach focuses on isolating the problem of operator selection from the search process, and proposing a learning mechanism that is conditioned on the decision space characteristics of the current solution. Furthermore, a fixed operator portfolio consisting of 4 destroy and 3 repair operators is used in their evaluation. In contrast, we propose a more robust operator selection system compatible with various operator portfolios of different sizes and train the system independently prior to integration with ALNS. Our approach also proposes the use of GNNs for scaling to large instances.

3 Methodology

3.1 Classic ALNS Algorithm

In ALNS [18], an initial solution is relaxed and re-optimised through iteratively employing a pair comprising a destroy operator $o_i^- \in \mathcal{D}$ and a repair operator $o_i^+ \in \mathcal{R}$ to form the new incumbent. The destroy scale d, which is randomly drawn or set as a hyper-parameter, describes the proportion of the solution that is destructed and reconstructed. In ALNS, the search can be divided into sequential segments, during which an initial score $\psi_i = 0$ is assigned to each operator (indexed by i) at the beginning and is increased by δ each time a new incumbent is formed using an operator pair that includes i. Depending on the

incumbent quality, the score is increased by δ_1 if the newly-found incumbent is a global best solution, δ_2 for a local best one, and δ_3 for an accepted yet worse local solution, where $\delta_1 > \delta_2 > \delta_3$. At the end of each segment, the cumulated score for each operator i and the number of times N_i it was selected are used to compute a weight w_i that estimates the operator's capability to find promising solutions. As shown in Equation (1), for each operator employed in the current segment K, its weight for the next segment $K+1$ is updated using a weighted average of the historical weight $w_{i,K}$ and its average performance in segment K.

$$w_{i,K+1} = \begin{cases} (1-\alpha_{\text{RW}}) \cdot w_{i,K} + \alpha_{\text{RW}} \cdot \frac{\psi_i}{N_i} & \text{if } \psi_i > 0, \\ w_{i,K} & \text{if } \psi_i = 0, \end{cases} \tag{1}$$

For each iteration within the segment, a pair of operators is selected using the RW selection algorithm with probabilities $w^-_{i,K} \big/ \sum_{j\in\mathcal{D}} w^-_{j,K}$ and $w^+_{i,K} \big/ \sum_{j\in\mathcal{R}} w^+_{j,K}$, where $w^{-/+}_{i,K}$ is the weight associated with each operator i in any given segment K. Initially, all operators are assigned the same score and therefore have the same selection probability. Once a new solution is formed, an ALNS acceptance mechanism, typically used in Simulated Annealing (SA), determines whether the newly-formed solution is accepted as the new incumbent to start the next iteration. The probabilistic acceptance mechanism helps to diversify the search and reduce the chance of becoming trapped in a non-promising local neighbourhood. The process continues until certain stopping criteria are met.

3.2 Operator Selection as a Markov Decision Process

Blueprint of our Approach. Our learning-based approach to improve the operator selection in ALNS consists, at a high level, of the following two steps. Firstly, we aim to isolate operator choice from the considerations of the SA process in ALNS, which introduces additional noise for navigating the solution space that may obscure the operators' contributions. To achieve this, we formulate operator selection for the COP as a standalone Markov Decision Process (MDP), in which an agent is given a limited budget of operators, and must learn to select those that lead to the best solutions. Secondly, the learned model is integrated into the ALNS loop and used to select operators in the SA process.

MDP Fundamentals. An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R)$. In each state $s \in \mathcal{S}$, the agent selects an action $a \in \mathcal{A}(s)$ out of a set of valid actions, receiving a reward r according to a reward function $R(s,a)$. Afterwards, the agent transitions to a new state s' that depends on $P(s'|s,a)$, which is the transition function that governs the environment dynamics. Interactions happen in episodes, each of which is a finite sequence of (s,a,r,s') pairs, until a terminal state is reached. Actions are selected by the agent through the *policy* $\pi(a|s)$ that completely specifies its behaviour. The state-action value function $Q(s,a)$ is the expected reward the agent receives by picking action a at a given state s, then following π.

MDP Formulation. We are given an undirected graph $G = (V, E)$ defined by the given COP and a feature matrix $\mathbf{X}$ in which each row contains information

about the node such as coordinates, demand, and distance. We formulate the MDP as below. A visualisation of an episode is shown in Figure 1.

- *States* $\mathcal{S}$: each state S_t is a tuple $(G, \mathbf{X}, J_t, C_t, \varphi, b_t)$, wherein the graph G and feature matrix $\mathbf{X}$ remain static. J_t is the set of *tours* that start and end at the depot, forming the solution at time t. The removal list $C_t = V \setminus J_t$ holds all the d nodes temporarily removed from the solution. φ indicates the *phase*: whether a destroy or repair operator is eligible to be applied. Finally, b_t indicates the operator pair budget available to the agent.
- *Actions* $\mathcal{A}$ involve the selection of an operator o_t, with those available defined as $\mathcal{D}$ if $\varphi = 1$ (i.e., we are in the destroy phase), and $\mathcal{R}$ otherwise.
- *Transitions* P apply the selected operator o_t to the current solution. Applying a destroy operator removes d nodes from J_t and places them in the removal list C_t. Using a repair operator reinserts the nodes from C_t into J_t, leaving C_t empty and the solution J_t complete, and decreases the operator pair budget by 1. Transitions are stochastic due to the inherent randomness of the operators.
- *Rewards* R are provided once the operator budget is exhausted and the improvement in solution quality can be assessed via an objective function F. Concretely, $R(S_t, A_t) = F(S_t) - F(S_0)$ if $b_t = 0$, and 0 otherwise.

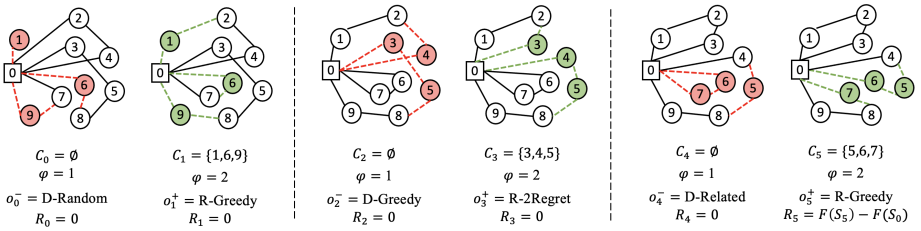

Fig. 1. Illustration of an MDP episode with budget $b = 3$ and destroy scale $d = 3$. The action spaces contain 3 destroy operators $\mathcal{D} = \{\text{Random, Greedy, Related}\}$ and 2 repairs $\mathcal{R} = \{\text{Greedy, 2Regret}\}$. The agent begins at state S_0 with $C_0 = \emptyset$ and routes $J_0 = \{[1], [2, 4], [3, 5, 8, 6], [7, 9]\}$, selecting operators o_0^- = Random and o_1^+ = Greedy to reach S_2. The episode continues until the budget is exhausted and the terminal state S_5 with routes $J_5 = \{[1, 2, 3], [4, 5, 6], [7, 8, 9]\}$ is reached. Finally, it receives a reward proportional to the improvement in solution quality.

3.3 Learning an Operator Selection Policy

Q-learning and relationship to Roulette Wheel update. Q-learning [27] is a model-free RL approach for solving MDPs that relies on estimating the state-action value function $Q(s, a)$, from which a policy π can be derived by acting greedily with respect to it. The agent's interactions with the environment generate (s, a, r, s') tuples, and its estimates are updated according to the rule

$$Q(s, a) \leftarrow (1 - \alpha_{\text{RL}}) \cdot Q(s, a) + \alpha_{\text{RL}} \cdot \left(r + \gamma \cdot \max_{a' \in \mathcal{A}(s')} Q(s', a') \right) \tag{2}$$

where α_{RL} is the learning rate, and γ trades immediate versus long-term rewards. Written in this form, comparing the Q-learning update in Equation (2) and the classic RW update in Equation (1), we notice that both use a weighted factor to balance two terms representing the historical and current estimates of performance. The key difference is that the Q-learning update is conditioned on the state and hence captures more information that may be used to select a relevant operator, while the RW update simply averages the gains of the operators irrespective of the context in which they were applied. Therefore, RW can be interpreted as a very rough approximation of the Q-learning update and, intuitively, using information about the state can allow us to obtain operator selection policies that perform at least as well. This means that Q-learning requires higher sample complexity. However, this was not an issue in practice, as we found a relatively low number of training steps suffices to reach a good policy.

Function Approximation and Graph Neural Networks. In problems with large state spaces, neural networks are commonly used to perform *function approximation* of the $Q(s, a)$ function. This helps to generalize between states that, while not being identical, share common characteristics and hence may lead to similar future rewards. The Deep Q-Network (DQN) algorithm [13], which uses this principle together with replay buffers and target networks, has been used for successfully approaching a variety of decision-making tasks.

In this work, we consider two possible neural network architectures. Firstly, we use a Multi-Layer Perceptron (MLP) formed of layers that apply a linear transformation of the inputs followed by a non-linear activation function. Despite their simplicity, MLPs are known to be universal function approximators. Secondly, we consider Graph Neural Network (GNN) architectures [20], that are explicitly designed to operate on graph-structured data. Such architectures compute an embedding for each node in the graph by iteratively aggregating the features of neighbouring nodes, resulting in node embeddings that encode both structural and feature-based information. A desirable characteristic of many GNN architectures is that their parametrization can be independent of the size of the input graph. Hence, they enable learning an approximation of the state-action value function on small instances and applying it directly on large instances – an appealing approach for COPs [3].

Integrating the model with ALNS. As mentioned above, the resulting learned policy acts greedily with respect to the learned state-action value function, always choosing the action with the highest expected cumulative reward. This might prove problematic once integrated within ALNS, given that, in principle, greediness may cause the search to become trapped in local optima. To instead obtain a *probabilistic* policy, we use a softmax function as shown in Equation (3), in which the temperature τ allows adjusting the level of greediness of the policy. Specifically, probabilities are uniform when $\tau \to \infty$, whereas the action with the highest expected reward has probability approaching 1 when $\tau \to 0$.

$$\pi_\tau(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a' \in \mathcal{A}(s)} \exp(Q(s,a')/\tau)} \tag{3}$$

3.4 Operators for ALNS

In the literature, operators are carefully tailored to fit different problem structures and features. Despite the large variety of operator designs, the mechanisms behind them are surprisingly similar to the first version of ALNS [18]. We conducted a thorough analysis of operators in the literature and have identified the following 3 classes: *random-based* destroy that randomly removes d nodes according to specific availability criteria, *greedy-based* destroy that removes the top-ranking d nodes with respect to a particular measure, and *related-based* destroy as an extension of Shaw's destroy [21] that removes the most similar d nodes according to a certain proximity value. Variations can include perturbations or using problem-specific features including distance, time, cost, workload, demand level, inventory level, removal gain, historical information, etc.

Barring a few *random-based* operators, almost all current repair operator designs are related to *greedy-based* mechanisms that insert each node at the position with the smallest cost. Variations can include a pre-sorting that changes the order of node insertions according to certain criteria, including global minimum insertion or smallest regret value. Others can have a noise factor that perturbs the insertion cost values, or use restrictions based on historical information.

4 Experiments

4.1 Experimental Setup

Problem Settings. In this work, we consider the Capacitated Vehicle Routing Problem (CVRP) with a single depot, a group of customer nodes and a number of homogeneous vehicles each visiting an individual group of customer nodes. The capacity restriction applies to the total carrying load of vehicles. Each customer node can only be visited once. We use the R, C and RC instances (random, clustered, and mixed random-clustered nodes) of the Solomon dataset [22] each containing a depot and 100 customers. We assign the vehicle capacity to be 200, and adjust it proportionally if scaling down the instance to fewer customers.

For the portfolio design, we identified 12 popular destroy operators from the ALNS literature that span the representative categories described in Section 3.4: the random-based variations *random node destroy* [18] and *random route destroy* [4], the greedy-based variations *worst-node removal* [18], *neighbourhood removal* [4] and *greedy route destroy* [9], and the related-based variations *proximity destroy* [4], *cluster destroy* [16], *node neighbourhood destroy* [4], *zone destroy* [5], *route neighbourhood destroy* [5], *pair destroy* [11] and *historical node-pair removal* [16]. The repair operator portfolio is comparatively smaller. We include the group of classic *greedy repair* [18] and *k-regret repair* [16] for $k = 2$.

Operator Selection Approaches. The proposed DQN agent is compared to the following approaches. As a baseline, we consider a uniform Random sampling (RAN) of operators. We also compare against the classic RW (CRW), which can only be used within ALNS since it requires information about the SA outcomes and search progress. To make the RW mechanism applicable in the MDP setting,

we make the following adaptations to obtain a method we call Learned RW (LRW). Firstly, in Equation (1), we replace the manually-defined operator scores ψ computed from the discretised δ with the continuous objective value F. We also adjust the reward feedback frequency from every operator pair in RW to every episode in the MDP. Preliminary experimental results suggested that the performance difference between the LRW and CRW is within 2% when applied in ALNS without any prior training.

Training and Evaluation Methodology. For each instance, we generate 3 distinct sets $\mathcal{J}^{\text{train}}, \mathcal{J}^{\text{validate}}, \mathcal{J}^{\text{test}}$ of 128 randomly initialized tours each. $\mathcal{J}^{\text{train}}$ is used by DQN and LRW for model training. $\mathcal{J}^{\text{validate}}$ is used for hyperparameter tuning and model selection. Finally, $\mathcal{J}^{\text{test}}$ is used to perform the final evaluation and obtain the reported results. There are two evaluation "modes": MDP-compatible agents can be evaluated in a standalone fashion given an operator budget (CRW is excluded), while all operators (including CRW) can be evaluated on the end ALNS task. Training and evaluation is repeated across 10 random seeds for all agents, which are used to compute confidence intervals.

DQN Architectures and Inputs. For the DQN, we consider MLP and GNN representations. The MLP has 256 units in the first hidden layer, with the subsequent layers having half the size. As a GNN, we opt for the GAT [26], which allows for flexible aggregation of neighbour features. We use 3 layers and a dimension of node embeddings equal to 32. Both use a learning rate of $\alpha_{\text{RL}} = 0.0005$ and are trained for $15 \cdot 10^3$ and $25 \cdot 10^3$ steps respectively. The DQN exploration rate ϵ is linearly decayed from 1 to 0.1 in the first 10% of steps, then remains fixed. The replay buffer size is equal to 20% of the number of steps. To obtain the inputs, we construct vectors $\tilde{\mathbf{x}}_t^i$ that concatenate the static instance-specific features $\mathbf{x}^i$ with time-dependant relevant information such as whether the node i is routed in a tour and the number of tours in J_t. For the MLP, we stack the vectors in a matrix $\widetilde{\mathbf{X}}_t$ as inputs, while for the GNN the node features are provided directly. Unless otherwise stated, we use a softmax temperature $\tau = 0.01$.

4.2 Experimental Results

Evaluating Agents within MDP Framework. In this experiment, we compare the cumulative rewards gained by the DQN with an MLP representation, LRW and RAN agents on the test set $\mathcal{J}^{\text{test}}$ after undergoing training. To make the training and evaluation processes less computationally intensive, we use the first 20 customer nodes and the depot from the Solomon R, C and RC instances. We define operator portfolios of different sizes ranging from 2 to 12 by sequentially adding the 12 destroy operators introduced above, together with the 2 repair operators. The destroy scale is fixed as $d = 4$ and the operator pair budget is $b = 10$, yielding MDP episodes of length 20.

Table 1 shows that the DQN agent is able to outperform competing methods as the size of $\mathcal{D}$ grows. When the destroy portfolio is smaller than 3, the DQN agent performs slightly worse due to the limited action space in which the impact of the selected actions is difficult to distinguish from chance. The DQN agent

Table 1. MDP evaluation results: cumulative rewards gained by the DQN, RAN and LRW agents with destroy portfolios $\mathcal{D}$ of different sizes. Higher is better.

	C-instance			R-instance			RC-instance		
$\lvert\mathcal{D}\rvert$	**DQN**	**RAN**	**LRW**	**DQN**	**RAN**	**LRW**	**DQN**	**RAN**	**LRW**
2	232.2±2.9	**252.2±3.6**	252.1±4.5	216.3±5	**222.9±3.7**	222.8±4.4	240.2±7.5	**259.4±3.4**	258.9±5.6
3	228.6±9.0	221.7±3.5	**245.9±6.0**	212.9±5.6	208.4±4.7	**215.1±3.8**	**236.1±6.1**	224.0±3.4	230.8±7.6
4	232.5±5.1	221.2±6.4	**240.3±5.8**	**220.2±3.1**	206.9±6.0	216.2±4.1	**241.2±5.8**	222.4±5.1	239.5±4.8
5	**328.8±2.4**	258.5±5.5	293.3±8.0	**330.9±4.2**	246.8±3.7	273.9±6.1	**331.1±2.9**	260.9±4.1	272.5±7.6
6	**329.9±4.2**	231.7±5.8	284.9±8.9	**329.5±3.3**	217.5±5.5	272.5±14.9	**329.5±2.6**	239.5±5.4	261.6±5.1
7	**328.5±2.8**	246.7±4.4	282.4±11.0	**330.5±3.8**	236.1±4.8	253.7±8.1	**331.7±2.9**	247.6±3.8	264.6±6.4
8	**329.5±3.9**	235.6±5.2	281.6±10.6	**330.9±3.6**	220.4±2.0	264.5±7.0	**333.7±3.3**	243.7±4.5	254.7±4.8
9	**330.7±3.1**	225.8±5.7	274.6±12.2	**328.9±4.6**	212.6±3.8	260.3±5.7	**332.4±3.7**	226.7±7.0	250.3±8.4
10	**330.2±4.5**	224.4±4.7	276.2±9.3	**330.3±3.3**	206.7±4.5	258.6±7.0	**331.0±2.6**	224.3±5.8	252.6±15.3
11	**330.3±2.9**	222.0±4.8	275.9±6.4	**327.2±8.0**	210.4±4.6	259.1±9.0	**326.1±15.2**	223.3±6.6	245.7±8.3
12	**361.8±0.2**	246.9±5.5	323.7±7.0	**404.2±0.6**	246.8±7.0	313.1±17.7	**354.9±4.1**	258.5±4.1	283.9±13.0
mean	**305.7±3.7**	235.2±5.0	275.5±8.2	**305.6±4.6**	221.4±4.6	255.4±8.0	**308.0±5.2**	239.1±4.8	255.9±7.9

also yields smaller confidence intervals and hence a steadier performance. As expected, the RAN agent fails to show a clear increase in rewards as the portfolio size grows. The LRW agent, although showing a certain improvement, performs significantly worse than the DQN. Two performance jumps in the DQN and LRW agents were observed: from size 4 to 5 and 11 to 12 for all 3 instances, the reason for which is the inclusion of a more efficient operator in the portfolio that suits the behaviour of a greedy-based agent.

ALNS Evaluation. Using the same experimental setup as above, we apply the operator selection approaches within ALNS with a fixed number of iterations. As shown in Table 2, the DQN agent yields the lowest objective values (best results) when used to perform operator selection in ALNS for portfolios larger than 5. Interestingly, LRW is able to perform substantially better than CRW due to having undergone training on a different dataset of solutions prior to being applied. Instead, the performance of CRW is indistinguishable from RAN in the setting across all the 3 instances.

Scaling to Larger Instances with GNN. In this experiment, we train the DQN with a GNN representation and the LRW on instances of size 20, then evaluate them in an MDP setting on instances of size up to 100. The operator budget is kept the same while the destroy scale is increased proportionally to the size, i.e., $d = n/5$. We use the largest destroy portfolio with $|\mathcal{D}| = 12$. As shown in the top half of Figure 2, the DQN+GNN agent outperforms the other methods, suggesting the strong generalization of the learned operator selection policies. A larger confidence interval is observed for the C instance, due to 1 model seed that generalizes poorly on $\mathcal{J}^{\text{test}}$ despite good performance on $\mathcal{J}^{\text{validate}}$.

Impact of Destroy Scale. Furthermore, we analyse the impact of the destroy scale on the agents' performances in the MDP setting, with a smaller scale implying the removal and reinsertion of a smaller proportion of nodes. We vary the destroy scale $d \in [2, 4, 6, 8, 10]$ with destroy portfolio $|\mathcal{D}| = 12$ on 20 nodes. Results are shown in the bottom half of Figure 2. The gap between the DQN and

Table 2. Evaluating operator selection approaches in ALNS with destroy portfolios $\mathcal{D}$ of different sizes. Values represent the average and best objective values found within a fixed number of iterations, using each approach to select operators. Lower is better.

C-inst	$\lvert\mathcal{D}\rvert$	2	3	4	5	6	7	8	9	10	11	12	mean
DQN	*avg*	316.28	314.96	311.18	**216.33**	**213.03**	**216.34**	**215.12**	**215.75**	**214.19**	**213.09**	**188.11**	**239.49**
	min	244.64	245.28	240.32	**154.06**	**149.29**	**153.05**	**151.87**	**153.04**	**150.7**	**149.99**	**146.71**	**176.27**
LRW	*avg*	**293.84**	**299.56**	**302.66**	248.31	259.06	261.17	266.59	264.65	265.62	264.78	224.49	268.25
	min	**209.8**	**211.62**	**207.6**	172.97	180.96	178.07	187.71	183.88	182.07	185.3	160.95	187.36
RAN	*avg*	293.5	313.98	315.05	284.31	299.86	294.57	299.64	303.58	312	305.74	286.08	300.76
	min	209.95	219.79	212.33	194.6	206.28	199.23	203.63	208.35	212.2	213.38	197.54	207.03
CRW	*avg*	293.94	314.56	314.28	285.53	299.51	295.77	302.67	307.64	311.32	308.53	286.75	301.86
	min	210.27	222.46	213.42	197.82	208.25	198.37	206.6	210.7	213.69	213.38	194.63	208.14
R-inst	$\lvert\mathcal{D}\rvert$	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**mean**
DQN	*avg*	330.37	334.09	331.24	**217.25**	**215.27**	**215.41**	**215.89**	**216.17**	**215.91**	**221.71**	**159.29**	**242.96**
	min	273.81	266.66	272.16	**144.97**	**143.56**	**144.11**	**144.19**	**144.58**	**144.49**	**153.64**	**108.07**	**176.39**
LRW	*avg*	**322.22**	**329.14**	**327.18**	269.97	265.26	286.74	281.67	286.63	284.94	274.17	238.46	287.85
	min	**246.19**	**254.83**	**243.38**	187.33	183.51	199.16	196.51	201.83	196.82	189.54	157.33	205.13
RAN	*avg*	323.25	337.03	337.68	298.76	317.84	310.1	321.11	321.76	327.19	321.3	294.45	319.13
	min	253.64	257.11	250.05	211.7	235.07	218.61	234.45	231.79	234.31	234.38	206.54	233.42
CRW	*avg*	322.71	333.59	336.81	298.38	320.5	308.7	318.42	321.34	327.17	324.96	296.05	318.97
	min	252.39	255.69	250.28	211.47	235.15	215.55	230.06	232.49	235.87	233.28	208.18	232.76
RC-inst	$\lvert\mathcal{D}\rvert$	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**mean**
DQN	*avg*	306.06	311.02	313.78	**216.84**	**218.75**	**218.16**	**217.66**	**217.48**	**218.12**	**224.56**	**201.81**	**242.2**
	min	228.16	235.04	240.93	**151.89**	**154.74**	**153.71**	**152.85**	**152.06**	**153.79**	**161.32**	**159.09**	**176.69**
LRW	*avg*	**294.94**	**314.93**	**310.02**	278.18	282.01	286.05	292.6	293.97	293.44	292.15	261.08	290.85
	min	**205.05**	**216.86**	**209.63**	187.78	189.92	190.45	197.61	196.75	197.62	196.21	172.7	196.42
RAN	*avg*	297.2	319.87	317.7	288.95	298.41	294.26	300.61	308.9	315.42	305.53	283.71	302.78
	min	207.17	222.85	209.95	193.42	198.33	195.88	200.04	207.6	210.9	204.25	185.29	203.24
CRW	*avg*	**294.94**	319.07	320.76	289.51	299.99	296.09	299.98	310.67	315.4	309.69	285.73	303.8
	min	**205.05**	221.27	213.01	195.39	200.23	196.03	202.8	207.05	211.47	205.59	186.24	204.01

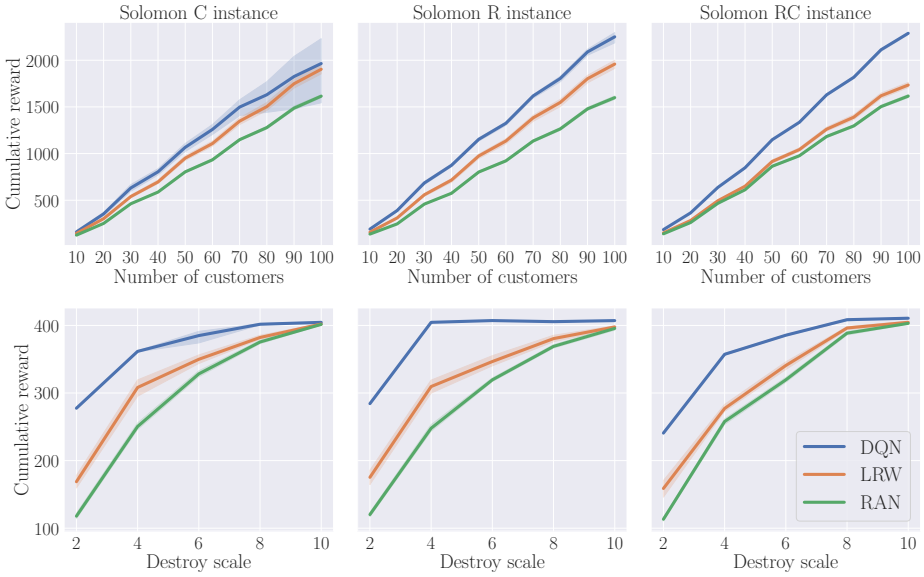

Fig. 2. Top: cumulative rewards for the DQN, LRW and RAN agents with GNN representation. Bottom: performance as a function of destroy scales. Higher is better.

other methods is largest for the smallest scale, suggesting that a careful selection of operators to remove the most expensive nodes contributes more significantly to better solution quality. In contrast, a larger destroy scale requires building up the solution from the ground, stressing the operators' reconstruction ability rather than the operator selection policy. When increasing the destroy scale, the cumulative rewards gained by different agents all converge to a similar level.

Impact of DQN Temperature. As discussed in Section 3.3, the temperature parameter τ controls the greediness of the resulting policy. Figure 3 shows the minimum and mean F obtained with ALNS as a function of $\tau \in \{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$, averaged over the 3 instance sets. Even though a probabilistic policy may be desirable in some ALNS scenarios, we find that performance generally degrades as the temperature increases. This suggests that, in the settings tested, the inherent stochasticity of the operators is sufficient to explore the search space without the need to combine different choices.

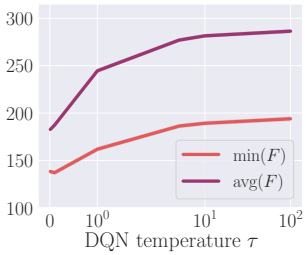

Fig. 3. Values of F when varying DQN temperature in ALNS. Lower is better.

5 Conclusions and Future Research

In this work, we have proposed an operator selection mechanism based on Deep Reinforcement Learning to enhance the performance of the ALNS metaheuristic. A key insight and contribution is the proposal of an operator selector that is conditioned on the decision space characteristics of the current solution. We have demonstrated its ability to outperform the classic Roulette Wheel and random operator selection, as well as the potential of using Graph Neural Networks to scale the model to large problem instances. Our results also highlight the impact of the operator portfolio size and the destroy scale on performance. Plans for future work involve applications to other combinatorial optimization problems.

Acknowledgements. This work was partially supported by The Alan Turing Institute under the Enrichment Scheme and the UK EPSRC grant EP/N510129/1.

References

1. Bai, R., Chen, X., Chen, Z.L., Cui, T., Gong, S., He, W., Jiang, X., Jin, H., Jin, J., Kendall, G., et al.: Analytics and machine learning in vehicle routing research. Int. J. Prod. Res. pp. 1–27 (2021)
2. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. In: ICLR Workshops (2016)
3. Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: a methodological tour d'horizon. Eur. J. Oper. Res. **290**(2), 405–421 (2021)
4. Demir, E., Bektaş, T., Laporte, G.: An adaptive large neighborhood search heuristic for the pollution-routing problem. Eur. J. Oper. Res. **223**(2), 346–359 (2012)

5. Emeç, U., Çatay, B., Bozkaya, B.: An adaptive large neighborhood search for an e-grocery delivery routing problem. Comput Oper Res. **69**, 109–125 (2016)
6. Falkner, J.K., Thyssens, D., Schmidt-Thieme, L.: Large neighborhood search based on neural construction heuristics. arXiv:2205.00772 (2022)
7. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. In: ECAI (2020)
8. Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A.M., Talbi, E.G.: Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. Eur. J. Oper. Res. **296**(2), 393–422 (2022)
9. Keskin, M., Çatay, B.: Partial recharge strategies for the electric vehicle routing problem with time windows. Transp. Res. Part C Emerg **65**, 111–127 (2016)
10. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: ICLR (2018)
11. Mancini, S.: A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. Transp. Res. Part C Emerg. **70**, 100–112 (2016)
12. Mara, S.T.W., Norcahyo, R., Jodiawan, P., Lusiantoro, L., Rifai, A.P.: A survey of adaptive large neighborhood search algorithms and applications. Comput. Oper. Res. p. 105903 (2022)
13. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
14. Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. In: NeurIPS (2018)
15. Oberweger, F., Raidl, G., Rönnberg, E., Huber, M.: A learning large neighborhood search for the staff rerostering problem. In: CPAIOR. pp. 300–317. Springer (2022)
16. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. Comput. Oper. Res. **34**(8), 2403–2435 (2007)
17. Reijnen, R., Zhang, Y., Lau, H.C., Bukhsh, Z.: Operator selection in adaptive large neighborhood search using deep reinforcement learning. arXiv:2211.00759 (2022)
18. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. **40**(4), 455–472 (2006)
19. Santini, A., Ropke, S., Hvattum, L.: A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. J. Heur **24**(5), 783–815 (2018)
20. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)
21. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: PPCP98. Lecture Notes, vol. 1520, pp. 417–431 (1999)
22. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research **35**(2), 254–265 (1987)
23. Syed, A.A., Akhnoukh, K., Kaltenhaeuser, B., Bogenberger, K.: Neural network based large neighborhood search algorithm for ride hailing services. In: EPIA Conference on Artificial Intelligence. pp. 584–595. Springer (2019)
24. Talbi, E.G.: Machine learning into metaheuristics: A survey and taxonomy. ACM Computing Surveys (CSUR) **54**(6), 1–32 (2021)
25. Turkeš, R., Sörensen, K., Hvattum, L.M.: Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search. Eur. J. Oper. Res. **292**(2), 423–442 (2021)
26. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)
27. Watkins, C., Dayan, P.: Q-learning. Machine Learning **8**(3-4), 279–292 (1992)