Operator Selection in Adaptive Large Neighborhood Search using Deep Reinforcement Learning

DOI:
10.48550/arXiv.2211.00759

Document status and date:
Published: 01/11/2022

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Operator Selection in Adaptive Large Neighborhood Search using Deep Reinforcement Learning

Robbert Reijnen [1], Yingqian Zhang [1], Hoong Chuin Lau [2], Zaharah Bukhsh [1]

[1] Eindhoven University of Technology [2] Singapore Management University
r.v.j.reijnen@tue.nl, yqzhang@tue.nl, hclau@smu.edu.sg, z.bukhsh@tue.nl

Abstract

Large Neighborhood Search (LNS) is a popular heuristic for solving combinatorial optimization problems. LNS iteratively explores the neighborhoods in solution spaces using destroy and repair operators. Determining the best operators for LNS to solve a problem at hand is a labor-intensive process. Hence, Adaptive Large Neighborhood Search (ALNS) has been proposed to adaptively select operators during the search process based on operator performances of the previous search iterations. Such an operator selection procedure is a heuristic, based on domain knowledge, which is ineffective with complex, large solution spaces. In this paper, we address the problem of selecting operators for each search iteration of ALNS as a sequential decision problem and propose a Deep Reinforcement Learning based method called Deep Reinforced Adaptive Large Neighborhood Search. As such, the proposed method aims to learn based on the state of the search which operation to select to obtain a high long-term reward, i.e., a good solution to the underlying optimization problem. The proposed method is evaluated on a time-dependent orienteering problem with stochastic weights and time windows. Results show that our approach effectively learns a strategy that adaptively selects operators for large neighborhood search, obtaining competitive results compared to a state-of-the-art machine learning approach while trained with much fewer observations on small-sized problem instances.

Introduction

Combinatorial Optimization Problems (COP) are widely studied problems in the operations research and computer science research communities. Practitioners aim in such problems to identify high-quality solutions in a large space of discrete decision variables. Given their computational complexity, often NP-hard, practical solution approaches typically rely on handcrafted heuristics that depend on trial and error. Such solutions are fast and capable of making decisions that are otherwise too expensive to compute, yet lack the guarantee of finding optimal solutions. On the other hand, owing to recent advancements, machine learning methodologies are increasingly being used for solving COP (Bengio, Lodi, and Prouvost 2021), potentially use data to learn better rules or policies. Still, defining end-to-end learning approaches that approximate a mapping function between the input and a solutions remains challenging; approaches typically face difficulties scaling to larger and more constrained or difficult problem variants.

Existing heuristic approaches have contributed significantly to solving these types of problems, yet are typically limited in the ability to scale to large sized problem instances and depend on expert knowledge. The Large Neighborhood Search (LNS) (Shaw 1998) is a popular heuristic that has shown considerable success in solving scheduling and transportation problems. This heuristic is based on the ruin-and-recreate principle of gradually improving solutions via continuous *destroy* and *repair* operators through heuristics (Schrimpf et al. 2000). This enables the search for better candidate solutions, traversing a promising search path. The LNS heuristic was correspondingly extended with Adaptive Large Neighborhood Search (ALNS) first proposed by Ropke and Pisinger (2006), allowing multiple destroy and repair operators to be used within the same search. In ALNS, each destroy and repair operator is assigned a weight that determines how to select operators in each iteration of the ALNS algorithmic search. These weights are adjusted dynamically based on operator performances; more successful operators are provided with a higher weight, and are therefore more likely to be selected in the next iteration of search.

A limitation of the weight-based operator selection procedure is that ALNS only includes past performances of operators for determining the operators' weight values and cannot take advantage of any short-term dependencies. Another limitation is the selection process required for determining which operators to include in ALNS. This typically requires initial experimenting, which is a time and computationally expensive procedure. To address these issues, machine learning approaches have been proposed that tune LNS approaches or directly learn how to destroy or repair a solution. Examples are the works of Hottung and Tierney (2019), in which a learning based LNS algorithm was proposed us a repair operator, the works of (Gao et al. 2020; Chen et al. 2020) for learning the destroy operator of the LNS algorithm to solve a Vehicle Routing Problem (VRP), and the work of Syed et al. (2019) that uses a neural network to directly modify solutions within the LNS algorithm. These approaches perform well on small-sized problem instances, yet, their ineffective scaling to solve large-scaled problems is a common issue.

As an alternative, we propose a new approach called Deep Reinforced Adaptive Large Neighborhood Search (DR-ALNS). We integrate in this approach a Deep Reinforcement Learning (DRL) decision-making policy in ALNS to learn operator selection. As such, we aim to learn to select the destroy and repair operators applied in the following search iteration within the ALNS framework. This enables fast reaction to changes in the search space and provides an atomized setup for experimentation, as we learn operator performances throughout the search process. We develop a problem-agnostic learning approach, i.e., the approach does not rely on specific information of the optimization problem and does not learn to output the decisions of the COP directly. This makes our proposed approach applicable to other combinatorial optimization problems with limited changes. Our proposed method is effective in selecting heuristics at each step of the search, providing a better heuristic search strategy than ALNS, which is provided with the same destroy and repair operators.

We evaluate our approach with the Time-Dependent Orienteering Problem with Stochastic Weights and Time Windows (TD-OPSWTW). The TD-OPSWTW was introduced in Verbeeck, Vansteenwegen, and Aghezzaf (2016) and is a variant of the widely studied Orienteering Problem (OP) (Gunawan, Lau, and Vansteenwegen 2016), where nodes are to be selected and visited in a particular order to maximize rewards while constrained by a time budget. In the stochastic variant of the problem, the time required to traverse from one location to another depends on a stochastic distribution. This, and time-dependent constraints make the problem difficult to solve with traditional optimization solvers. For this reason, this problem was selected for the IJCAI AI4TSP competition (Bliek et al. 2022).

We summarize the contributions of our work as follows:

- We propose the first Deep Reinforcement Learning based operator selection approach for Adaptive Large Neighborhood Search. The experiment results show our approach outperforms ALNS in terms of solution quality and solution consistency. Furthermore, it gives a comparable performance to a state-of-the-art end-to-end DRL approach, which requires longer training time than ours.
- Our approach is generalized to different scales of the problem; as such, we can train a model on the cheaper to compute instances and deploy it for finding solutions to the computational expensive problem instances.

Background and Related Work

Large Neighborhood Search (LNS) (Shaw 1998) is a general metaheuristic framework for solving COPs, including transportation and scheduling problems (Pisinger and Ropke 2010). This framework searches for better solutions by repeatedly removing parts from a solution and reinserting them at a more profitable location. This search is performed by 'destroy' and 'repair' heuristics, where the destroy heuristic deletes a part of a solution to a certain problem and the repair heuristic aims to repair this 'destroyed' solution. More specifically, a solution x is partly broken, after which it is repaired to derive the next solution x^t. Together, they define a solution Neighborhood $\mathcal{N}(x)$, comprised of solution candidates that can be accessed from a solution x. The original LNS algorithm only accepts new solutions in the iterative search that improve the cost function. Other works have also used a Simulated Annealing-based acceptance criterion, in which a new found solution x^t can also be accepted with a probability $e^{-\left(c(x^t)-c(x)\right)/T}$ if $c(x) \leq c(x^t)$ (Schrimpf et al. 2000). Here, T is a temperature which gradually decreases after each iteration, accepting more deteriorating solutions at the start of the search. A complete overview of commonly used acceptance criteria is given in Santini, Ropke, and Hvattum (2018). LNS is continued until a termination condition is reached. Commonly used termination criteria are the setting of a maximum number of search iterations to be performed, a time limit, or a predefined number of iterations without improvement of the best-found solution.

Adaptive Large Neighborhood Search (ALNS) (Ropke and Pisinger 2006) is an extension to the LNS heuristics, using a set of destroy operators $d \in \Omega^-$ and repair operators $r \in \Omega^+$ in the search. Each destroy and repair operator is assigned a weight that controls how often it is attempted in the search. Weights are denoted as $\rho^- \in \mathbb{R}^{|\Omega^-|}$ for destroy operators and $\rho^+ \in \mathbb{R}^{|\Omega^+|}$ for repair operators and initialized with equal initial values. The weights are dynamically updated after an iteration of ALNS is completed, based on the quality of the solutions obtained. The updated weight values are calculated as: $\rho_a^- = \lambda\rho_a^- + (1-\lambda)\psi$ and $\rho_b^+ = \lambda\rho_b^+ + (1-\lambda)\psi$, where λ is a decay parameter that controls the sensitivity of the weight changes and ψ as a parameter score for based on the obtained solution values. With the weights are probabilities calculated for selecting destroy and repair operators: the probability ϕ_j^- of selecting destroy operator j is calculated as $\frac{\rho_j^-}{\sum_{k=1}^{|\Omega^-|}\rho_k^-}$, and the probability of selecting an repair operator is calculated in the same way. The pseudocode of ALNS is formulated in Algorithm 1.

Algorithm 1: Adaptive Large Neighborhood Search (ALNS) (Pisinger and Ropke 2010)

```
Input: feasible solution $x$; $x_{best} = x$;
$\rho^- = (1, \ldots, 1)$; $\rho^+ = (1, \ldots, 1)$;
while Stopping criteria is not met do
    select destroy and repair operators $d \in \Omega^-$ and
      $r \in \Omega^+$ using $\rho^-$ and $\rho^+$;
    $x^t = r(d(x))$;
    if accept $(x^t, x)$ then
        $x = x^t$;
    if $c(x^t) < c(x_{best})$ then
        $x_{best} = x^t$;
    update $\rho^-$ and $\rho^+$;
return $x_{best}$
```

Machine learning for improving LNS Many machine learning (ML) based methods have been developed to solve routing problems (e.g., Kool, Van Hoof, and Welling (2018); Joe and Lau (2020); da Costa et al. (2021)). In the past years, several works use ML to improve the iterative search algorithm by, for example, effectively searching the neighborhood at each iteration or modifying a solution within LNS and ALNS framework with learning. Hottung and Tierney (2019) use neural networks with attention mechanisms as a repair operator within the LNS framework to solve routing problems. Syed et al. (2019) also uses a neural network in LNS with input data composed of domain-specific features (e.g., regret values). Both Gao et al. (2020) and Chen et al. (2020) use a learning-based approach as a destroy operator within LNS to solve a vehicle routing problem. For this, Gao et al. apply the effect of graph topology on the solution and used a graph attention network with edge-embedding as an encoder, and Chen et al. uses the proximate policy optimization (PPO) algorithm to train a hierarchical recursive graph convolution network (GCN) as a destroy operator. Both approaches are used to repair solutions by inserting nodes into a destroyed solution to formulate a feasible solution. Song et al. (2020) learns a neighborhood selection policy using imitation learning and reinforcement learning (RL). They use this to decompose the set of integer variables into subsets of fixed sizes. Correspondingly, they define each subset as a sub-problem. Sonnerat et al. (2021) train a neural network using imitation learning to select the variables to be removed from a solution. Wu et al. (2021) proposes an approach in which the action of RL is used to select the variables to be replaced. With this, RL is used to determine how to destroy a solution.

These end-to-end approaches approximate a mapping function between the input and a solution. However, such approaches typically face difficulties scaling to larger and more constrained or difficult problem variants as the size of state and action space increases enormously with the size of problem instances, requiring much larger data samples, and hence much longer training time, to learn good policies. Furthermore, most of the existing machine learning approaches aim to construct solutions to the optimization problem at hand, hence, are difficult to be directly applied to other variants of the problems. In this paper, we propose a new approach that uses Deep Reinforcement Learning within Adaptive Large Neighborhood Search to mitigate these shortcomings.

Deep Reinforced Adaptive Large Neighborhood Search (DR-ALNS)

We propose a new approach for improving Adaptive Large Neighborhood Search called Deep Reinforced Adaptive Large Neighborhood Search (DR-ALNS). This approach uses Deep Reinforcement Learning within the Adaptive Large Neighborhood Search framework for selecting destroy and repair operators to be attempted in the search. Where other proposed works that use Deep Reinforcement Learning for improving LNS heuristics are typically designed for specific optimization problems, we aim to leverage DRL in a generalize fashion. The pseudocode of the proposed approach, with its training alogrithm is defined in 2.

Markov Decision Process Formulation

To apply Deep Reinforcement learning within the ALNS framework, the learning problem is modeled as a sequential decision making problem, such that the agent can learn to interact with the problem environment by performing sequences of actions for finding solutions. A widely used mathematical framework to do this modeling is a Markov Decision Process (MDP). Such an MDP framework is defined as a tuple $\langle S, A, R, P \rangle$, where S represents the set of states, A the set of actions, R the reward function, and P the state transition probability function. The state transition occurs after the execution of an action by the agent, such that the state S_t of the environment at time step t transforms to S_{t+1} at time step $t + 1$, and the agent receives a reward according to the reward function R. The goal of a DRL agent is to learn a policy function π_s that maps states s_t to actions a_t that maximize the expected sum of future rewards. We formulate the MDP to create an environment where agents can learn to select which destroy and repair heuristics to apply in every iteration of the Large Neighborhood Search procedure. For this, we formulate the state space S, action space A and reward function R as follows:

Algorithm 2: Deep Reinforced Adaptive Large Neighborhood Search (DR-ALNS)

```
Training:
Input: number of training steps $M$;
step = 0;
while step $< M$ do
    Initialize problem instance;
    Initialize feasible solution $x$;
    $x_{best} = x$;
    Initialize initial state $s_{t=0}$;
    while Stopping criteria is not met do
        select action $a$ with policy $\pi_\theta$ based on state $s_t$;
        select destroy operator $d$ and repair operator $r$ based on action $a$;
        $x^t = r(d(x))$
        if accept $(x^t, x)$ then
            $x = x^t$;
        if $c(x^t) < c(x_{best})$ then
            $x_{best} = x^t$;
        step = step + 1
        update state $s_t$ and receive reward $r_t$
    Update policy $\pi_\theta$ based on $s_t$

Deployment:
Input: operator selection policy $\pi_\theta$;
feasible solution $x$; $x_{best} = x$
while Stopping criteria is not met do
    select destroy and repair operators $d \in \Omega^-$ and $r \in \Omega^+$ with policy $\pi_\theta$ based on state $s$;
    $x^t = r(d(x))$;
    if accept $(x^t, x)$ then
        $x = x^t$;
    if $c(x^t) < c(x_{best})$ then
        $x_{best} = x^t$;
return $x_{best}$
```

State space S is formulated as a one dimensional vector as information regarding the state of the search. With this, we provide the agent with information that can be used for selecting the best possible actions for a certain search iteration. The state space consists of 8 problem-agnostic features which are shown in Table 1.

Table 1: State features

Feature	Description
Improvement	Current solution improved: 0 or 1
Cost_difference_best	% difference between objective values of current and best solutions (-1 if current is ≤ 0)
Is_current_best	Current equal to best (0 or 1)
Temperature	Current temperature of Simulated Annealing acceptance criterion
Stagnation count	Number of iterations without improving the best found solution
Iteration	Iteration number
Current_accepted	Accepted: 0 or 1
Current_improved	Accepted & better than previous solution: 0 or 1

Action space A consists of all possible combinations of destroy and repair operators that can be applied in an iteration of search. E.g., an ALNS formulation containing 4 destroy operators and 3 repair operators will have action space of $3 \times 4 = 12$ actions. The size of the actions space is therefore dependent on the number of both the destroy and repair operators that can be imposed on a solution.

Reward function R is formulated for the learning to select actions (operators) based on the state S of the search process. To shape the reward function, we make use of the original scoring function of ALNS defined in (Ropke and Pisinger 2006), formulated as follow:

$$R_t = \begin{cases} 5, & \text{if } f\left(x^t\right) > f\left(x^{\text{best}}\right) \\ 3, & \text{if } f\left(x^t\right) > f(x) \\ 1, & \text{if accept } \left(x^t, x\right) \\ 0, & \text{otherwise,} \end{cases}$$

in which x is the current solution, x^t is the new generated solution and x^{best} the best found solution. With this reward function we aim to encourage the reinforcement learning agent to gradually find better solutions, by improving either current solutions or the best found solution. By also providing a small reward for solutions that are accepted by the accepted criteria, we also reward situations in which the agent has found slightly worse solutions that are accepted by the acceptance criteria. We argue that these solutions also contribute to the search procedure, and can potentially encourage the diversification of the operators used.

State Transition Function P. In this work, the agent has no prior knowledge of this transition function and learns it by interacting with the environment.

With the formulated MDP, we provide a DRL Learning agent with an environment for learning to select operators for DR-ALNS. The state space S and reward function R are defined in a problem-agnostic manner, i.e., they do not rely on COP-specific information. Therefore, formulated MDP can be applied for learning operator selection for DR-ALNS to solve any COP. Therefore, to use the approach to solve a COP, practitioners are only required to define the 'destroy' and 'repair' operators. The product of the two types of operators is correspondingly used as the action space A within the formulated MDP.

The TD-OPSWTW problem

We apply our approach to solve the Time-dependent Orienteering Problem with Stochastic Weights and Time Windows (TD-OPSWTW), which is used in the IJCAI AI4TSP competition (Bliek et al. 2022). This problem is a more realistic version of the classical Travelling Salesman Problem (TSP), in which travel costs between locations are unknown, the salesman has a limited travel time capacity, and where customers can only be visited within specific time windows. Each customer, represented as a node, has a certain prize which represents the importance of its visit and can only be visited in a predefined time-window. Due to the time budget of the salesman, not all nodes can be visited. Therefore, the objective of the problem is to collect the most prizes in the network as possible, while still respecting the time budget constraint. The main challenge of the problem is to deal with the stochastic travel times between locations, that are only revealed once the salesman is deployed in the network.

More formally, the problem is defined as a set of n nodes with x and y coordinates of the nodes in a 2D space that makes a complete graph. The stochastic travel times $t_{i,j} \in \mathbb{R}, \forall i, j \in \{1, \ldots, n\}$ between node i and j are are computed by multiplying the Euclidean distance $d_{i,j}$ by a noise term η that follows a discrete uniform distribution $\mathcal{U}\{1, 100\}$ normalized by a scaling factor $\beta = 100$, i.e., $\tau_{i,j} = d_{i,j} \frac{\eta}{\beta}$, where $t_{i,j}$ are samples from $\tau_{i,j}$. With this, the travel time between i and j may differ in different samples. Each node i has a time window denoted by a lower bound $\{l_i \in \mathbb{N}\}_{i=1}^n$ and an upper bound $\{h_i \in \mathbb{N}\}_{i=1}^n$ and is allocated with a prize $\{p_i \in \mathbb{R}\}_{i=1}^n$. This prize can be collected when the node is visited within its time window. Each instance has a maximum tour length T, determining the maximum time the salesman can spend to collect prizes. Solutions to the problem must respect the time windows and the maximum tour time. Each violation of the constraint is treated with a penalties $\{e_i \in \mathbb{R}\}_{i=1}^n$. All solutions (tours) that take longer than T are penalized by $e_i = -n$, incurred at the node i at which the violation occurred. Moreover, each time window violation incurs a penalty of $e_i = -1$ at the current node i.

Destroy and Repair Operators Formulation

Adaptive Large Neighborhood Search has been applied to solve variants of orienteering problems, e.g., in Santini (2019); Hammami, Rekik, and Coelho (2020); Yahiaoui, Moukrim, and Serairi (2019); Roozbeh, Hearne, and Pahlevani (2020). Based on these works, we have defined four destroy operators and three repair operators for solving the TD-OPSWTW problem addressed in this work. The destroy operators defined are based on *random remove* and *random* *edge remove*. In random remove, n customers are selected uniformly from a given solution $(0, i_1, \ldots, i_k, 0)$ and removed from the solution. With random sequence remove, we aim to address the risk that similar solutions are obtained with an repair operator after removal with the random remove destroy operator. Therefore, the random sequence remove destroy operator is formulated to destroy sequences of customers from a solution. We formulate two variants for both destroy operators, a 'modest' and 'severe' one, to define the number customers to be removed from a solution. The variants remove a random number between 0 - 25% and 20 - 40% of the customers from a solution.

The repair operators are defined to increase the total reward obtained by a solution for the problem by inserting more customers to be visited in a solution. To do so, we identified three repair operators, that insert new customers in the 'destroyed' solution: *random distance repair*, *random prize repair* and *random ratio repair*. In random distance repair, we randomly select a number of customers to add to the solution and, according to a randomly generated sequence, insert them into the solution at their least expensive position (in terms of distance). Random prize repair and random ratio repair work according to the same principle, yet insert nodes sequentially at the position where the total accumulated rewards are maximized, or the ratio between reward and additional distance traveled is optimized. All destroy and repair operators are compatible, so every repair operator can 'repair' any solution that has been destroyed by any destroy operator. Because of this, the four destroy operators and three repair operators define an action space of $3 \times 4 = 12$ actions that can be applied to an iteration of the search.

Experiments

We evaluate the performance of the proposed DR-ALNS by solving the TD-OPSWTW problem. We use the publicly available problem instances and simulator that are published by the IJCAI AI4TSP competition organizers at https://github.com/paulorocosta/ai-for-tsp-competition. We follow the same training and evaluation protocol of the competition (Bliek et al. 2022). The implementation of our approach will be made public.

Model training

For the agent training, we select the Proximal Policy Optimization (PPO) algorithm of Schulman et al. (2017) for learning the operator-selection policy π_θ of our proposed approach. The PPO is a policy-gradient method that uses a probability ratio between two policies to maximize improvement without the risk of performance collapse. As such, it utilizes a clipping function to define how far away a new policy is allowed to deviate from the old policy, preventing catastrophic forgetting. Given its computational inexpensiveness, ease of implementation, and effectiveness for learning a wide range of sequential decision-making tasks, it is regarded as one of the most successful DRL algorithms and is widely used by practitioners.

The training was performed for 250.000 steps with 100 iterations of search. We train 3 models for 20, 50 and 100 instances respectably, each trained with 100 different problem instances. The training was performed on a Processor Intel(R) Core(TM) i7-6920HQ CPU @ 2.90GHz with 8.0GB of RAM with ten parallel environments, which took around 2, 7.5, and 32 hours of training for instances of sizes 20, 50, and 100, respectively. The model parameters selected for training are described in Schulman et al. (2017). The training traces are displayed in Figure 1, showing the mean episodic reward obtained during the training.

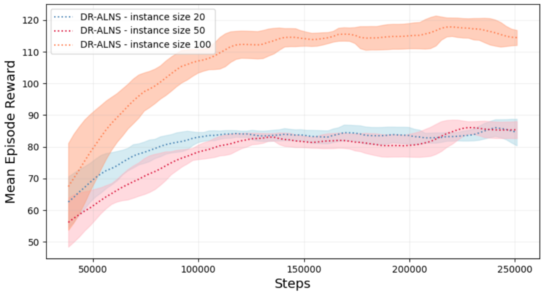

Figure 1: Rolling mean and standard deviation of training episode reward over time

Baselines

Two baselines are selected for comparing the performance of the proposed DR-ALNS approach to solve the problem at hand: *ALNS* and *Rise Up*. ALNS is implemented with the same 'destroy' and 'repair' operators as the DR-ALNS algorithm. We use Simulated Annealing as an acceptance criterion with '1' as starting temperature, and a linear decay step of 1/100 is used for cooling down. The temperature is bounded at 0.25. For updating the weights, we use convex combinations of the current weights and the update parameters. These update parameters are equal to the reward scheme of the DR-ALNS approach. The Rise Up is the winning solution of the AI4TSP competition. The proposed algorithm is based on POMO reinforcement learning of Kwon et al. (2020) to turn a deep neural network into an end-to-end approach for constructing solutions directly. The network architecture consists of an encoder and a decoder neural network that exploits symmetries to encourage exploration during learning. For solving the TD-OPSWTW, the authors provide problem-specific information to the input for each node, including information on instance prizes, time constraint information, travel times, and the embedding of the current node in a route and of the depot. Masking is used to prevent invalid and/or infeasible actions from being taken. This approach is used to train a DRL policy per problem instance size, training for several days until full convergence on a single Tesla V100 GPU. Subsequently, an Efficient Active Search (EAS) procedure was used to fine-tune the learned policies for each instance (Hottung, Kwon, and Tierney 2021). This EAS uses reinforcement learning to fine-tune a policy to a single test instance. As such, an individualized policy is tuned for each instance. Finally, Monte-Carlo roll-outs are used for constructing the final solutions by sampling actions with the learned policies.

Experimental Results

Performances of the tested algorithms are evaluated on various problem instances of TD-OPSWTW, as described in (Bliek et al. 2022). The instances used in our test contain respectively 20, 50, and 100 nodes. We implemented the ALNS algorithm. Following the competition protocol, both the ALNS and DR-ALNS algorithms were run fifty times with different random seeds for the repair operators, and are initialized with an empty route as initial solution with a solution quality of 0.00 (no prizes nor penalties are obtained). The stopping criteria used by both algorithms is set to 100 iterations of search. Results from the 'Rise Up' approach are obtained by directly evaluating their submitted solutions to the AI4TSP competition. The best solutions obtained in each run are shown in Tables 2-4, together with standard deviations, medians, and the lower and upper quartiles.

Solution quality. Table 2 shows algorithm performances for solving the smallest problem instances. These instances are of equal size to the instances used for training the proposed DR-ALNS algorithm. From the table can be observed that ALNS is capable of finding most of the best solutions for these instances. However, on the contrary, ALNS also faces issues finding solutions of higher quality than the initialized solution, as for all instances, expect 'instance0105', ALNS cannot find a better that the initial one for 25% of the found solutions. This can potentially be caused by accepting a solution of very poor quality or by continuous re-visiting of the initialized solution. The Rise-Up algorithm performs very stably, finding identical solutions that obtain the same rewards. DR-ALNS performs on average the best, finding the best average algorithm performance for four out of five instances. The results highlight that DRL can effectively be used to learn a stable operator selection policy within the Adaptive Large Neighborhood algorithm.

Table 2: Algorithm performances to find solutions for small problem instances (20 nodes) of TD-OPSWTW

Model	mean	std	min	25%	50%	75%	max
	instance0101						
ALNS	1,15	0,94	0,00	0,00	1,93	1,93	**1,93**
DR-ALNS	**1,93**	0,03	1,70	1,93	1,93	1,93	**1,93**
Rise Up	**1,93**	0,00	1,93	1,93	1,93	1,93	**1,93**
	instance0102						
ALNS	1,78	1,54	0,00	0,00	2,74	3,06	**3,88**
DR-ALNS	**3,58**	0,20	3,06	3,46	3,74	3,74	3,74
Rise Up	3,06	0,00	3,06	3,06	3,06	3,06	3,06
	instance0103						
ALNS	2,62	1,94	0,00	0,00	3,56	4,14	**5,03**
DR-ALNS	**4,67**	0,21	4,41	4,51	4,51	4,89	4,89
Rise Up	4,14	0,00	4,14	4,14	4,14	4,14	4,14
	instance0104						
ALNS	1,46	1,33	0,00	0,00	2,12	2,69	**2,95**
DR-ALNS	**2,94**	0,05	2,69	2,95	2,95	2,95	**2,95**
Rise Up	2,41	0,00	2,41	2,41	2,41	2,41	2,41
	instance0105						
ALNS	**9,07**	0,61	6,56	8,79	9,21	9,54	9,81
DR-ALNS	8,73	0,23	8,44	8,55	8,62	8,88	9,54
Rise Up	8,88	0,00	8,88	8,88	8,88	8,88	8,88

Tables 3 and 4 show the results of the algorithms to solve larger problem instances with 50 and 100 nodes. We also apply the DR-ALNS approach that has been trained on the smallest problem instances (DR-ALNS (20)) to solve larger instances. With this, we aim to evaluate the extent the approach can scale to large-sized instances without training with these larger-sized instances that are computationally more expensive due to increased repair options that need to be evaluated. The solutions obtained with the Rise Up algorithm are obtained with the DRL that has been trained on instances of the same size. From both tables, the proposed DR-ALNS approach outperforms the ALNS method in terms of the average solutions found, finding a better average solution for 9 of 10 instances. This can be explained by the ALNS approach's dependence on random seeds in the repair operators to find better solutions than the initialized solution, potentially being stuck to local optima. Also, from the large problem instance results, the proposed DR-ALNS approach can find better solutions to the large problem instances compared with ALNS, also when trained with the smallest instances. This highlights the proposed approach's effective ability to use learning for selecting operators for the search; ALNS is likely to have not finished its search, requiring more iterations of search to find potentially better solutions. This can potentially be explained by the absence of prior operator performance knowledge when ALNS is initialized or the inability of the ALNS approach to deal with short-term dependencies when the search state is suddenly changed. This also explains why the DR-ALNS approach trained with small problem instances (DR-ALNS (20)) performs better than ALNS to solve these instances, as it can make use of its prior experiences to select destroy and repair operators effectively. Overall, the performance gap with the Rise Up algorithm is found to decrease when instance sizes get larger, and the Rise up solution find better solutions and better average solutions for most of the largest problem instances. This shows that the proposed DR-ALNS approach is less capable to deal with the largest sized instances, potentially requiring more iterations of search or different destroy operators to meet this performances.

Convergence performance of ALNS and DR-ALNS is shown in Figure 2. From the figure can be observed that after training, DR-ALNS finds better solutions in fewer iterations of search, as compared to the ALNS algorithm. This emphasizes the ability of DR-ALNS to make use prior leanings to effectively select destroy and repair operators to be attempted in the search, which results in a much faster convergence than the traditional ALNS.

Computation time. For Rise Up, the authors trained one separate policy model for each of the considered problem sizes. Each model is trained for several days until full convergence on a single Tesla V100 GPU (Bliek et al. 2022). In comparison, the policies trained for the DR-ALNS have been trained on a Processor Intel(R) Core(TM) i7-6920HQ CPU @ 2.90GHz with 8.0GB of RAM, which took around 2, 7.5, and 32 hours of training for instances of sizes 20, 50, and 100. After training, the required time for DR-ALNS to find solutions is 4, 30 and 120 seconds, for the different instance sizes. This highlights the computational efficiency of the proposed algorithm to solve COPs, requiring only a fraction of the computational budget used by the end-to-end approach for the training.

Table 3: Algorithm performances to find solutions for medium problem instances (50 nodes) of TD-OPSWTW

Model	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
	instance0351						
ALNS	2,82	2,62	0,00	0,00	3,53	5,25	**6,46**
DR-ALNS (20)	4,83	1,65	0,00	5,17	5,37	5,55	5,78
DR-ALNS	4,94	1,35	0,00	4,88	5,33	5,59	5,88
Rise Up	**5,78**	0,00	5,78	5,78	5,78	5,78	5,78
	instance0352						
ALNS	3,61	3,32	0,00	0,00	4,78	6,66	8,83
DR-ALNS (20)	7,20	1,39	1,61	6,54	7,25	8,07	**10,64**
DR-ALNS	6,39	2,57	0,00	5,24	7,09	7,95	9,85
Rise Up	**9,30**	0,00	9,30	9,30	9,30	9,30	9,30
	instance0353						
ALNS	1,39	1,26	0,00	0,00	1,46	2,65	**3,47**
DR-ALNS (20)	2,25	0,88	0,00	2,36	2,58	2,69	3,11
DR-ALNS	2,47	0,73	0,00	2,47	2,59	2,87	3,11
Rise Up	**2,77**	0,00	2,77	2,77	2,77	2,77	2,77
	instance0354						
ALNS	3,07	2,49	0,00	0,00	4,56	5,11	**6,15**
DR-ALNS (20)	5,07	0,40	4,03	4,76	5,04	5,35	5,87
DR-ALNS	5,20	1,15	0,00	5,12	5,41	5,69	6,11
Rise Up	**5,41**	0,00	5,41	5,41	5,41	5,41	5,41
	instance0355						
ALNS	18,34	3,54	0,00	17,45	19,03	20,40	22,42
DR-ALNS (20)	19,27	1,50	16,61	18,13	18,89	20,30	**22,82**
DR-ALNS	20,16	2,44	11,96	18,52	20,72	22,04	**22,82**
Rise Up	**22,45**	0,00	22,45	22,45	22,45	22,45	22,45

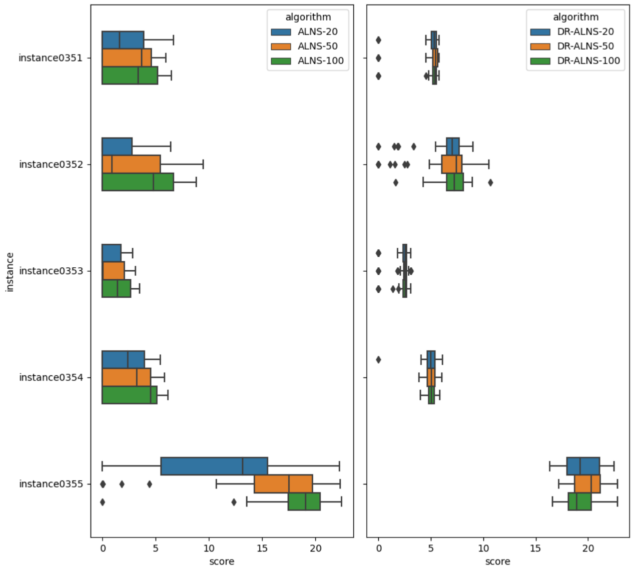

Figure 2: Performance comparison of ALNS and DR-ALNS provided with 20, 50 and 100 iterations of search

Table 4: Algorithm performances to find solutions for large problem instances (100 nodes) of TD-OPSWTW

Model	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
	instance0601						
ALNS	6,25	3,69	0,00	4,20	7,66	8,68	12,15
DR-ALNS (20)	9,40	1,20	6,70	8,67	9,27	9,95	12,47
DR-ALNS	10,01	2,07	0,00	9,03	9,79	11,10	13,60
Rise Up	**12,87**	5,34	-4,41	15,57	15,59	15,80	**15,82**
	instance0602						
ALNS	5,98	5,33	0,00	0,00	7,16	10,62	15,21
DR-ALNS (20)	11,02	2,68	0,00	10,30	11,54	12,53	14,41
DR-ALNS	9,66	4,48	0,00	8,89	10,95	11,89	15,97
Rise Up	**16,92**	0,00	16,92	16,92	16,92	16,92	**16,92**
	instance0603						
ALNS	4,03	3,10	0,00	0,00	5,26	6,52	8,39
DR-ALNS (20)	5,6	2,94	0,00	5,74	6,81	7,47	9,40
DR-ALNS	6,45	2,84	0,00	6,45	7,48	8,09	**9,65**
Rise Up	**9,12**	0,00	9,12	9,12	9,12	9,12	9,12
	instance0604						
ALNS	4,24	3,05	0,00	0,00	5,02	6,87	8,47
DR-ALNS (20)	6,95	0,66	5,26	6,49	6,97	7,29	8,41
DR-ALNS	7,01	1,72	0,00	6,71	7,21	7,79	9,63
Rise Up	**10,17**	0,00	10,17	10,17	10,17	10,17	**10,17**
	instance0605						
ALNS	4,49	3,35	0,00	0,00	5,84	7,00	9,44
DR-ALNS (20)	7,22	1,43	1,73	6,51	7,17	7,94	10,22
DR-ALNS	6,94	2,72	0,00	5,88	7,67	8,48	11,12
Rise Up	**12,30**	0,11	12,01	12,28	12,28	12,41	**12,41**

Conclusion

We propose DR-ALNS, a Deep Reinforcement Learning based operator-selection approach for the Adaptive Large Neighborhood Search heuristic. The proposed method has been applied to solve the Time-Dependent Orienteering problem with Stochastic Weights and Time Windows. We compare our method with the classical ALNS method, in which operators are selected based on recent performances, and with an end-to-end learning approach that has shown to be very successful in solving the problem at hand. Results show that the proposed method is very effective in solving problems of the smallest size, finding superior solutions compared to the selected baselines. Also, the proposed method is very capable of selecting operators within the ALNS framework to solve larger problem instances, consistently outperforming the weight-based ALNS, despite the limited training time with small-sized problem instances. Further, our approach can find competitive solutions to the end-to-end machine learning solution, despite being trained with limited observations and without problem-specific features included in the training. This makes the approach potentially interesting to apply to other problems.

Acknowledgments

This work is supported by the Luxembourg National Research Fund (FNR) (15706426).

References

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

Bliek, L.; da Costa, P.; Afshar, R. R.; Zhang, Y.; Catshoek, T.; Vos, D.; Verwer, S.; Schmitt-Ulms, F.; Hottung, A.; Shah, T.; et al. 2022. The First AI4TSP Competition: Learning to Solve Stochastic Routing Problems. *arXiv preprint arXiv:2201.10453*.

Chen, M.; Gao, L.; Chen, Q.; and Liu, Z. 2020. Dynamic partial removal: A neural network heuristic for large neighborhood search. *arXiv preprint arXiv:2005.09330*.

da Costa, P.; Rhuggenaath, J.; Zhang, Y.; Akcay, A.; and Kaymak, U. 2021. Learning 2-Opt Heuristics for Routing Problems via Deep Reinforcement Learning. *SN Computer Science*, 2(5): 1–16.

Gao, L.; Chen, M.; Chen, Q.; Luo, G.; Zhu, N.; and Liu, Z. 2020. Learn to design the heuristics for vehicle routing problem. *arXiv preprint arXiv:2002.08539*.

Gunawan, A.; Lau, H. C.; and Vansteenwegen, P. 2016. Orienteering Problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2): 315–332.

Hammami, F.; Rekik, M.; and Coelho, L. C. 2020. A hybrid adaptive large neighborhood search heuristic for the team orienteering problem. *Computers & Operations Research*, 123: 105034.

Hottung, A.; Kwon, Y.-D.; and Tierney, K. 2021. Efficient active search for combinatorial optimization problems. *arXiv preprint arXiv:2106.05126*.

Hottung, A.; and Tierney, K. 2019. Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*.

Joe, W.; and Lau, H. C. 2020. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In *Proceedings of the international Conference on Automated Planning and Scheduling*, volume 30, 394–402.

Kool, W.; Van Hoof, H.; and Welling, M. 2018. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*.

Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 21188–21198.

Pisinger, D.; and Ropke, S. 2010. Large neighborhood search. In *Handbook of metaheuristics*, 399–419. Springer.

Roozbeh, I.; Hearne, J. W.; and Pahlevani, D. 2020. A solution approach to the orienteering problem with time windows and synchronisation constraints. *Heliyon*, 6(6): e04202.

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

Santini, A. 2019. An adaptive large neighbourhood search algorithm for the orienteering problem. *Expert Systems with Applications*, 123: 154–167.

Santini, A.; Ropke, S.; and Hvattum, L. M. 2018. A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic. *Journal of Heuristics*, 24(5): 783–815.

Schrimpf, G.; Schneider, J.; Stamm-Wilbrandt, H.; and Dueck, G. 2000. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2): 139–171.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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

Song, J.; Yue, Y.; Dilkina, B.; et al. 2020. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33: 20012–20023.

Sonnerat, N.; Wang, P.; Ktena, I.; Bartunov, S.; and Nair, V. 2021. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*.

Syed, A. A.; Akhnoukh, K.; Kaltenhaeuser, B.; and Bogenberger, K. 2019. Neural network based large neighborhood search algorithm for ride hailing services. In *EPIA Conference on Artificial Intelligence*, 584–595. Springer.

Verbeeck, C.; Vansteenwegen, P.; and Aghezzaf, E.-H. 2016. Solving the stochastic time-dependent orienteering problem with time windows. *European Journal of Operational Research*, 255(3): 699–718.

Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2021. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34: 30075–30087.

Yahiaoui, A.-E.; Moukrim, A.; and Serairi, M. 2019. The clustered team orienteering problem. *Computers & Operations Research*, 111: 386–399.