RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark

Federico Berto[*1], **Chuanbo Hua**[*1], **Junyoung Park**[*1], **Laurin Luttmann**[*2], **Yining Ma**[3], **Fanchen Bu**[1], **Jiarui Wang**[4], **Haoran Ye**[5], **Minsu Kim**[1], **Sanghyeok Choi**[1], **Nayeli Gast Zepeda**[6], **André Hottung**[6], **Jianan Zhou**[3], **Jieyi Bi**[3], **Yu Hu**[7], **Fei Liu**[8], **Hyeonah Kim**[1], **Jiwoo Son**[15], **Haeyeon Kim**[1], **Davide Angioni**[9], **Wouter Kool**[10], **Zhiguang Cao**[11], **Qingfu Zhang**[8], **Joungho Kim**[1], **Jie Zhang**[3], **Kijung Shin**[1], **Cathy Wu**[12], **Sungsoo Ahn**[13], **Guojie Song**[5] **Changhyun Kwon**[1,14], **Kevin Tierney**[6], **Lin Xie**[2,14], **Jinkyoo Park**[1,15]

[1]KAIST, [2]Leuphana University, [3]Nanyang Technological University, [4]Southeastern University, [5]Peking University, [6]Bielefeld University, [7]Soochow University, [8]City University of Hong Kong, [9]University of Brescia, [10]ORTEC, [11]Singapore Management University, [12]MIT, [13]POSTECH, [14]Twente University, [15]OMELET, AI4CO

Abstract

Deep reinforcement learning (RL) has recently shown significant benefits in solving combinatorial optimization (CO) problems, reducing reliance on domain expertise, and improving computational efficiency. However, the field lacks a unified benchmark for easy development and standardized comparison of algorithms across diverse CO problems. To fill this gap, we introduce RL4CO, a unified and extensive benchmark with in-depth library coverage of 23 state-of-the-art methods and more than 20 CO problems. Built on efficient software libraries and best practices in implementation, RL4CO features modularized implementation and flexible configuration of diverse RL algorithms, neural network architectures, inference techniques, and environments. RL4CO allows researchers to seamlessly navigate existing successes and develop their unique designs, facilitating the entire research process by decoupling science from heavy engineering. We also provide extensive benchmark studies to inspire new insights and future work. RL4CO has attracted numerous researchers in the community and is open-sourced at https://github.com/ai4co/rl4co.

1 Introduction

Combinatorial optimization (CO) focuses on finding optimal solutions for problems with discrete variables and has broad applications, including vehicle routing [89, 60], scheduling [128], and hardware device placement [53]. Given that the combinatorial space expands exponentially and exhibits NP-hard characteristics, the operations research (OR) community has traditionally tackled these challenges through the development of mathematical programming algorithms [35] and handcrafted heuristics [27]. Despite their success, these methods still face significant limitations: mathematical programming struggles with scaling, while handcrafted heuristics require significant domain-specific adjustments for different CO problems.

[*]Equal contribution.

Work made with contributions from the AI4CO open research community.

Submitted to the 38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks. Do not distribute.

Recently, to address these limitations, neural combinatorial optimization (NCO) [7] has emerged. It employs deep neural networks to automate the problem-solving process and significantly reduces the computation demands and the need for domain expertise. Recent NCO works mainly leverage the reinforcement learning (RL) paradigm, making significant strides in improving exploration efficiency [62, 54], relaxing the needs of obtaining optimal solutions, and extending to various CO tasks [128, 89, 60, 53]. Although supervised learning (SL) methods [29] are shown to be effective in NCO, they require the availability of high-quality solutions, which is unrealistic for large instances or theoretically hard problems. Therefore, we focus on the widespread RL paradigm in this paper.

Despite the growing popularity and advancements in using reinforcement learning for solving combinatorial optimization, there remains a lack of a unified benchmark for analyzing past works under consistent implementations and conditions. The absence of a standardized benchmark hinders NCO researchers' efforts to make impactful advancements and leverage existing successes, as it becomes challenging to determine the superiority of one method over another. Moreover, the significance of NCO lies in its potential for generalizability across multiple problems without extensive problem-specific knowledge. Variations in implementation can make it difficult for new researchers to engage with the NCO community, and inconsistent comparisons obstruct straightforward performance evaluations. These issues pose significant challenges and underscore the need for a comprehensive benchmark to streamline research and foster consistent progress.

Contributions. To bridge this gap, we introduce RL4CO, the first comprehensive benchmark with multiple baselines, environments, and boilerplate from the literature, all implemented in a *modular*, *flexible*, *accelerated*, and *unified* manner. Our aim is to facilitate the entire research process for the NCO community with the following key contributions: 1) **Simplifying development** through modularizing 27 environments and 23 existing baseline models, allowing for flexible and automated combinations for effortless testing, switching, and achieving state-of-the-art performance; 2) **Enhancing the training and testing efficiency** through the customized unified pipeline tailored for the NCO community based on advanced libraries such as TorchRL [15], PyTorch Lightning [31], Hydra [123], and TensorDict [15]; 3) **Standardizing evaluation** to ensure fair and comprehensive comparisons, enabling researchers to automatically test a broader range of problems from diverse distributions and gather valuable insights using our testbed. Overall, RL4CO eliminates the need for repetitive heavy engineering in the NCO community and fosters seamless future development by building on existing successes, enabling advanced innovation and progress in the field.

2 Related Works

Neural Combinatorial Optimization. Neural combinatorial optimization (NCO) utilizes machine learning techniques to automatically develop novel heuristics for solving NP-hard CO problems. We classify the majority of NCO research from the following perspectives: 1) *Learning Paragiams*: researchers have employed supervised learning [115, 108, 29, 75] to approximate optimal solutions to CO instances. Further research leverages reinforcement learning [6, 89, 60, 62], and unsupervised learning [39, 84] to ease the difficulty of obtaining (near-)optimal solutions. 2) *Models*: various deep learning architectures such as recurrent neural networks [115, 22, 68], graph neural networks [48, 84], Transformers [60, 62], diffusion models [108], and GFlowNets [129, 56] have been employed. 3) *Problems*: NCO has demonstrated great success in various problems, including vehicle routing problems (VRPs) (e.g., traveling salesman problem and capacitated VRP), scheduling problems (e.g., job shop scheduling problems [128]), hardware device placement [53], and graph-based CO problems (e.g., maximum independent set [23, 2] and maximum cut [129]). 4) *Heuristic Types*: generally, the learned heuristics can be categorized as *constructive* in an autoregressive [60] or non-autoregressive [48] way, and *improvement* heuristics, which leverage traditional heuristics [120, 80] and meta-heuristics [105]. We refer to Bengio et al. [7] for a comprehensive survey. In this paper, we focus on the reinforcement learning paradigm due to its effectiveness and flexibility. Notably, the proposed RL4CO is versatile to support most combinations of models, problems and heuristic types, making it an apt library and benchmark for future research in NCO.

Table 1: Comparison of libraries in reinforcement learning for combinatorial optimization.

Library	Environments #	Baselines[†] #	Hardware Acceleration	Availability	Modular Baselines	Open Community
ORL [4]	3	1	×	×	×	×
OR-Gym [42]	9	-	×	✓	×	×
Graph-Env [12]	2	-	×	✓	×	×
RLOR [116]	2	2	×	✓	✓	×
RoutingArena [111]	1	8	✓	×	×	×
Jumanji [14]	22	3	✓	✓	×	×
RL4CO (ours)	27[‡]	23	✓	✓	✓	✓

[†] We consider as *baselines* ad-hoc network architectures (i.e., policies) and RL algorithms from the literature.
[‡] We also consider the possible 16 combinations of environments generated by the unified Multi-Task VRP, as they have been historically considered separate environments in the NCO literature.

Related Benchmark Libraries. Despite the variety of general-purpose RL software libraries [18, 70, 96, 119, 24, 33, 81], there is a lack of a unified and extensive benchmark for CO problems. Balaji et al. [4] propose an RL benchmark for Operations Research (OR) with a PPO baseline [100]; Hubbs et al. [42], Biagioni et al. [12] provide a collection of OR environments. Wan et al. [116] propose a general-purpose library for OR, and benchmarks the canonical TSP and CVRP environments. However, a major downside of the above libraries is that they cannot be massively parallelized due to their reliance on the OpenAI Gym API, which can only run on CPU, unlike RL4CO, which is based on the TorchRL [15], a recent official PyTorch [92] library for RL that enables hardware-accelerated execution of both environments and algorithms. Prouvost et al. [94] introduces a library specialized for CO problems that work in combination with traditional MILP [71] solvers. We also mention Routing Arena [111], whose scope is different from RL4CO, namely, comparing NCO and classical solvers only for the CVRP. The most related work is Jumanji [14], which provides a variety of CO environments written in JAX [16] that can be hardware-accelerated alongside an actor-critic baseline. While Jumanji is an RL environment suite, RL4CO is a full-stack library that integrates environments, policies, RL algorithms under a unified framework.

3 RL4CO: Taxonomy

We describe the RL4CO taxonomy, categorizing components into *Environments, Policies,* and *RL Algorithms*. Then we translate the taxonomy to implementation in § 4.

Environments. Given a CO problem instance $\boldsymbol{x}$, we formulate the solution-generating procedure as a Markov Decision Process (MDP) characterized by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ as follows. **State** $\mathcal{S}$ is the space of states that represent the given problem $\boldsymbol{x}$ and the current partial solution being updated in the MDP. **Action** $\mathcal{A}$ is the action space, which includes all feasible actions a_t that can be taken at each step t. **State Transition** $\mathcal{T}$ is the deterministic state transition function $s_{t+1} = \mathcal{T}(s_t, a_t)$ that updates a state s_t to the next state s_{t+1}. **Reward** $\mathcal{R}$ is the reward function $\mathcal{R}(s_t, a_t)$ representing the immediate reward received after taking action a_t in state s_t. Finally, $\gamma \in [0, 1]$ is a discount factor that determines the importance of future rewards. Since the state transition is deterministic, we represent the solution for a problem $\boldsymbol{x}$ as a sequence of T actions $\boldsymbol{a} = (a_1, \dots, a_T)$. Then the total return $\sum_{t=1}^{T} \mathcal{R}(s_t, a_t)$ translates to the negative cost function of the CO problem.

Policies. The policies can be categorized into constructive policies, which generate a solution from scratch, and improvement policies, which refine an existing solution.

Constructive policies. A policy π is used to construct a solution from scratch for a given problem instance $\boldsymbol{x}$. It can be further categorized into autoregressive (AR) and non-autoregressive (NAR) policies. An AR policy is composed by an encoder f that maps the instance $\boldsymbol{x}$ into an embedding space $\boldsymbol{h} = f(\boldsymbol{x})$ and by a decoder g that iteratively determines a sequence of actions $\boldsymbol{a}$ as follows:

$$a_t \sim g(a_t | a_{t-1}, ..., a_0, s_t, \boldsymbol{h}), \quad \pi(\boldsymbol{a}|\boldsymbol{x}) \triangleq \prod_{t=1}^{T-1} g(a_t | a_{t-1}, \dots, a_0, s_t, \boldsymbol{h}). \tag{1}$$

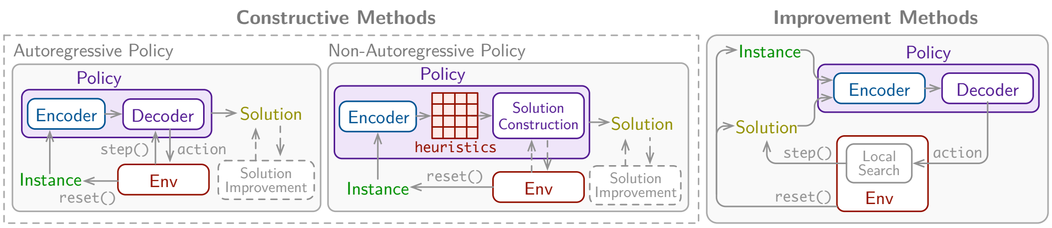

Figure 1: Overview of different types of policies and their modularization in RL4CO.

A NAR policy encodes a problem $\boldsymbol{x}$ into a heuristic $\mathcal{H} = f(\boldsymbol{x}) \in \mathbb{R}^N_+$, where N is the number of possible assignments across all decision variables. Each number in $\mathcal{H}$ represents a (unnormalized) probability of a particular assignment. To obtain a solution $\boldsymbol{a}$ from $\mathcal{H}$, one can sample a sequence of assignments from $\mathcal{H}$ while dynamically masking infeasible assignments to meet problem-specific constraints. It can also guide a search process, e.g., Ant Colony Optimization [28, 125, 56], or be incorporated into hybrid frameworks [127]. Here, the heuristic helps identify promising transitions and improve the efficiency of finding an optimal or near-optimal solution.

Improvement policies. A policy can be used for improving an initial solution $\boldsymbol{a}^0 = (a^0_0, \ldots, a^0_{T-1})$ into another one potentially with higher quality, which can be formulated as follows:

$$\boldsymbol{a}^k \sim g(\boldsymbol{a}^0, \boldsymbol{h}), \quad \pi(\boldsymbol{a}^K|\boldsymbol{a}^0, \boldsymbol{x}) \triangleq \prod_{k=1}^{K-1} g(\boldsymbol{a}^k|\boldsymbol{a}^{k-1}, ..., \boldsymbol{a}^0, \boldsymbol{h}), \tag{2}$$

where $\boldsymbol{a}^k$ is the k-th updated solution and K is the budget for number of improvements. This process allows continuous refinement for a long time to enhance the solution quality.

RL Algorithms. The RL objective is to learn a policy π that maximizes the expected cumulative reward (or equivalently minimizes the cost) over the distribution of problem instances:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x})} \left[\mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{x})} \left[\sum_{t=0}^{T-1} \gamma^t \mathcal{R}(s_t, a_t) \right] \right], \tag{3}$$

where θ is the set of parameters of π and $P(\boldsymbol{x})$ is the distribution of problem instances. Eq. (3) can be solved using algorithms such as variations of REINFORCE [109], Advantage Actor-Critic (A2C) methods [59], or Proximal Policy Optimization (PPO) [100]. These algorithms are employed to train the policy network π, by transforming the maximization problem in Eq. (3) into a minimization problem involving a loss function, which is then optimized using gradient descent algorithms. For instance, the REINFORCE loss function gradient is given by:

$$\nabla_\theta \mathcal{L}_a(\theta|\boldsymbol{x}) = \mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{x})} \left[(R(\boldsymbol{a}, \boldsymbol{x}) - b(\boldsymbol{x})) \nabla_\theta \log \pi(\boldsymbol{a}|\boldsymbol{x}) \right], \tag{4}$$

where $b(\cdot)$ is a baseline function used to stabilize training and reduce gradient variance. We also distinguish between two types of RL (pre)training: 1) *inductive* and 2) *transductive* RL. In inductive RL, the focus is on learning patterns from the training dataset to generalize to new instances, thus amortizing the inference procedure. Conversely, transductive RL (or test-time optimization) optimizes parameters during testing on target instances. Typically, a policy π is trained using inductive RL, followed by transductive RL for test-time optimization.

4 RL4CO: Library Structure

RL4CO is a unified reinforcement learning (RL) for Combinatorial Optimization (CO) library that aims to provide a *modular*, *flexible*, and *unified* code base for training and evaluating RL for CO methods with extensive benchmarking capabilities on various settings. As shown in Fig. 2, RL4CO decouples the major components of an RL pipeline, prioritizing their reusability in the implementation. Following also the taxonomy of § 3, the main components are: (§ 4.1) Environments, (§ 4.2) Policies, (§ 4.3) RL algorithms, (§ 4.4) Utilities, and (§ 4.5) Environments & Baselines Zoo.

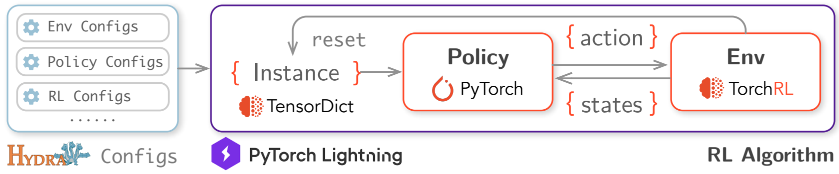

Figure 2: Overview of the RL4CO pipeline: from configurations to training a policy.

4.1 Environments

Environments in RL4CO fully specify the CO problems and their logic. They are based on the `RL4COEnvBase` class that extends from the `EnvBase` in TorchRL [15]. A modular `generator` can be provided to the environment. The generator provides CO instances to the environment, and different generators can be used to generate different data distributions. Static instance data and dynamic variables, such as the current state s_t, current solution $\boldsymbol{a}^k$ for improvement environments, policy actions a_t, rewards, and additional information are passed in a *stateless* fashion in a `TensorDict` [86], that we call `td`, through the environment `reset` and `step` functions. Additionally, our environment API contains several functions, such as `render`, `check_solution_validity`, `select_start_nodes` (i.e., for POMO-based optimization [62]) and optional API as `local_search` solution improvement.

It is noteworthy that RL4CO enhances the efficiency of environments when compared to vanilla TorchRL, by overriding and optimizing some methods in TorchRL `EnvBase`. For instance, our new `step` method brings a decrease of up to 50% in latency and halves the memory impact by avoiding saving duplicate components in the stateless `TensorDict`.

4.2 Policies

Policies in RL4CO are subclasses of PyTorch's `nn.Module` and contain the encoding-decoding logic and neural network parameters θ. Different policies in the RL4CO "zoo" can inherit from metaclasses like `ConstructivePolicy` or `ImprovementPolicy`. We modularize components to process raw features into the embedding space via a parametrized function ϕ_ω, called *feature embeddings*. 1) *Node Embeddings* ϕ_n: transform m_n node features of instances $\boldsymbol{x}$ from the feature space to the embedding space h, i.e., $[B, N, m_n] \rightarrow [B, N, h]$. 2) *Edge Embeddings* ϕ_e: transform m_e edge features of instances $\boldsymbol{x}$ from the feature space to the embedding space h, i.e., $[B, E, m_e] \rightarrow [B, E, h]$, where E is the number of edges. 3) *Context Embeddings* ϕ_c: capture contextual information by transforming m_c context features from the current decoding step s_t from the feature space to the embedding space h, i.e., $[B, m_c] \rightarrow [B, h]$, for nodes or edges. Overall, Fig. 3 illustrates a generic constructive AR policy in RL4CO, where the feature embeddings are applied similarly to other types of policies. Embeddings can be automatically selected by RL4CO at runtime by simply passing the `env_name` to the policy. Additionally, we allow for granular control of any higher-level policy component independently, such as encoders and decoders.

4.3 RL Algorithms

RL algorithms in RL4CO define the process that takes the `Environment` with its problem instances and the `Policy` to optimize its parameters θ. The parent class of algorithms is the `RL4COLitModule`, inheriting from PyTorch Lightning's `pl.LightningModule` [31]. This allows for granular support of various methods including the `[train, val, test]_step`, automatic logging with several logging services such as Wandb via `log_metrics`, automatic optimizer configuration via `configure_optimizers` and several useful callbacks for RL methods such as `on_train_epoch_end`. RL algorithms are additionally attached to an `RL4COTrainer`, a wrapper we made with additional optimizations around `pl.Trainer`. This module seamlessly supports features of modern training pipelines, including logging, checkpoint management, mixed-precision training, various hardware acceleration supports (e.g., CPU, GPU, TPU, and Apple Silicon), and multi-device hardware accelerator in distributed settings [69]. For instance, using mixed-precision

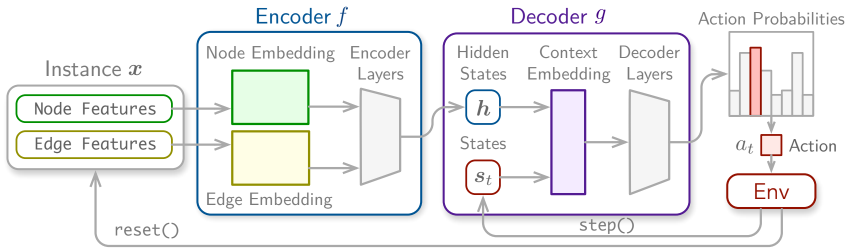

Figure 3: Overview of modularized RL4CO policies. Any component such as the encoder/decoder structure and feature embeddings can be replaced and thus the model is adaptable to various new environments.

training significantly decreases training time without sacrificing much convergence and enables us to leverage recent routines, e.g., FlashAttention [26, 25], which we investigate in Appendix.

4.4 Utilities

Configuration Management. Optionally, but usefully, we adopt `Hydra` [123], an open-source Python framework that enables hierarchical config management, making it easier to manage complex configurations and experiments with different settings as shown in Appendix. Hydra additionally allows for automatically parsing parameters (un-)defined in configs - i.e., `python run.py experiment=routing/pomo env=cvrp env.generator_params.num_loc=50` launches an experiment defined under `routing/pomo` and changes the environment to CVRP with 50 locations.

Decoding Schemes. Decoding schemes handle the logic of model logits z by applying preprocessing, such as masking of infeasible actions and/or additional techniques to select better actions during training and testing. We implement the model and problem-agnostic decoding schemes under the `DecodingStrategy` class in the RL4CO codebase that can be easily reused: 1) *Greedy*, which selects the action with the highest probability; 2) *Sampling*, which samples `n_samples` solutions from the current masked probability distribution of the policy, incorporating sampling strategies like 2.a) Softmax Temperature τ, 2.b) top-k sampling [61], and 2.c) top-p (or Nucleus) sampling [38] (more details in Appendix); 3) *Multistart*, which enforces diverse starting actions as demonstrated in POMO [62], such as starting from different cities in the Traveling Salesman Problem (TSP) with N nodes; 4) *Augmentation*, which applies transformations to instances, such as random rotations and flipping in Euclidean problems [55], to create an augmented set of problems.

Documentation, Tutorials, and Testing. We release extensive documentation to make it as accessible as possible for both newcomers and experts. RL4CO can be easily installed by running `pip install rl4co` with open-source code available at https://github.com/ai4co/rl4co. Several tutorials and examples are also available under the `examples/` folder. We thoroughly test our library via continuous integration on multiple Python versions and operating systems. The following code snippet shows minimalistic code that can train a model in a few lines:

```python
from rl4co.envs.routing import TSPEnv, TSPGenerator
from rl4co.models import AttentionModelPolicy, POMO
from rl4co.utils import RL4COTrainer
# Instantiate generator and environment
generator = TSPGenerator(num_loc=50, loc_distribution="uniform")
env = TSPEnv(generator)
# Create policy and RL model
policy = AttentionModelPolicy(env_name=env.name, num_encoder_layers=6)
model = POMO(env, policy, batch_size=64)
# Instantiate Trainer and fit
trainer = RL4COTrainer(max_epochs=10, accelerator="gpu", precision="16-mixed")
trainer.fit(model)
```

4.5 Environments & Baselines Zoo

Environments. We include benchmarking from the following environments, divided into four areas. 1) **Routing**: Traveling Salesman Problem (TSP) [65], Capacitated Vehicle Routing Problem (CVRP) [13], Orienteering Problem (OP) [64, 21], Prize Collecting TSP (PCTSP) [5], Pickup and Delivery Problem (PDP) [50, 99] and Multi-Task VRP (MTVRP) [72, 131, 9] (which modularizes with 16 problem variants including the basic VRPTW, OVRP, VRPB, VRPL and VRPs with their constraint combinations); 2) **Scheduling**: Flexible Job Shop Scheduling Problem (FJSSP) [17], Job Shop Scheduling Problem (JSSP) [97] and Flow Shop Scheduling Problem (FJSP); 3) **Electronic Design Automation**: multiple Decap Placement Problem (mDPP) [53]; 4) **Graph**: Facility Location Problem (FLP) [30] and Max Cover Problem (MCP) [51].

Baseline Zoo. Given that several works contribute to both new policies and new RL algorithm variations, we list the papers we reproduce. For 1) **Constructive AR** methods, we include the Attention Model (AM) [60], Ptr-Net [115], POMO [62], MatNet [63], HAM [67], SymNCO [55], PolyNet [41], MTPOMO [72], MVMoE [131], L2D [128], HGNN [106] and DevFormer [53]. For 2) **Constructive NAR** methods, we benchmark Ant Colony Optimization-based DeepACO [125] and GFACS [56] as well as the hybrid NAR/AR GLOP [127]. 3) **Improvement methods** include DACT [78], N2S [79] and NeuOpt [80]. We also include 4) **General-purpose RL** algorithm from the literature, including REINFORCE [109] with various baselines, Advantage Actor-Critic (A2C) [59] and Proximal Policy Optimization (PPO) [100] that can be readily be combined with any policy. Finally, we include 5) **Active search** (i.e., Transductive RL) methods AS [6] and EAS [40].

5 Benchmarking Study

We perform several benchmarking studies with our unified RL4CO library. Given the limited space, we invite the reader to check out the Appendix for supplementary material.

5.1 Flexibility and Modularity

Changing policy components. The integration of many state-of-the-art methods in RL4CO from the NCO field in a modular framework makes it easy to implement and improve upon state-of-the-art neural solvers for complex CO problems with only a few lines of code and improve upon them.[2] We demonstrate this in Table 2 for the FJSSP by gradually replacing or adding elements to the original SotA policy [106]. First, replacing the HGNN encoder with the more expressive MatNet encoder [63] already improves the average makespan by around 7%. Further improvements can be achieved by replacing the MLP decoder with the Pointer mechanism in the AM decoder [60] with gaps to BKS around 3× lower compared to the original policy in Song et al. [106] even with greedy performance.

Table 2: Solutions obtained with RL4CO for the FJSSP with different model configurations.

Encoder / Decoder		FJSSP 10×5	FJSSP 20×5
HGNN + MLP (g.) [106]	Obj.	111.82	211.21
	Gap	15.8%	12.1%
MatNet + MLP (g.)	Obj.	103.91	197.92
	Gap	7.6%	5.0%
MatNet + Pointer (g.)	Obj.	101.17	196.3
	Gap	4.8%	4.2%
MatNet + Pointer (s. x128)	Obj.	98.31	192.02
	Gap	1.8%	1.9%

5.2 Constructive Policies

Mind Your Baseline. In on-policy RL, which is often employed in RL4CO due to fast reward function evaluations, several different REINFORCE baselines have been proposed to improve the performance. We benchmark several RL algorithms training constructive policies for routing problems of node size 50, whose underlying architecture is based on the encoder-decoder Attention Model [60] and whose main difference lies in how the REINFORCE baseline is calculated (we additionally train the AM with PPO as further reference). For a fair comparison, we run all baselines

[2] The different model configurations shown here can be obtained by simply changing the Hydra configuration file like the one shown in Appendix.

in controlled settings with the same number of optimization steps and report results in Table 3. We note that A2C generally underperforms other baselines. Such performance can be attributed to the fact that since in routing problems, the rewards are sparse (i.e., can only be calculated upon solving an entire problem), estimating the value of an entire instance $\boldsymbol{x}$ is inherently a challenging task. Interestingly, while POMO [62], which takes as a baseline the shared baseline of all routes forcing each starting node to be different, may work well as baselines for problems in which near-optimal solutions can be constructed from any node (e.g., TSP), this may not be true for other problems such as the Orienteering Problem (OP): the reason is that in OP only a *subset* of nodes should be selected in an optimal solution, while several states will be discarded. Hence, forcing the policy to select all of them makes up for a poor baseline. We remark that while SymNCO (whose shared baseline involves symmetric rotations and flips) [55] may perform well in Euclidean problems, this is not applicable in non-Euclidean CO, including asymmetric routing problems and scheduling. We found similar trends regarding actor-critic methods as A2C and PPO in the EDA mDPP problem [53], which we report in Appendix. Namely, a greedy rollout baseline [60] can do better than value-based methods due to the challenging task of instance value estimation.

Table 3: Optimality Gap obtained via greedy decoding.

Method	TSP	CVRP	OP	PCTSP	PDP
A2C	2.22	7.09	8.64	14.96	10.02
AM-Rollout	1.41	5.30	4.40	2.46	9.88
POMO	0.89	3.99	14.26	11.61	10.64
Sym-NCO	0.47	4.61	3.09	2.12	7.73
AM-PPO	0.92	4.60	3.05	2.45	8.31

Decoding Schemes. The solution quality of NCO solvers often shows significant improvements in performance to different decoding schemes, even with the exact NCO solvers. We evaluate the trained solver with different decoding schemes and settings as shown in Fig. 4.

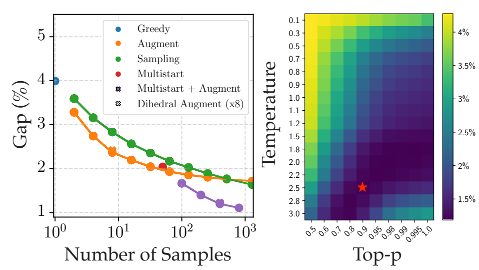

Figure 4: Decoding schemes study of POMO on CVRP50. [Left]: Pareto front of decoding schemes by the number of samples; [Right]: performance of sampling with different temperatures τ and p values for top-p sampling.

Generalization. Using RL4CO, we can easily evaluate the generalization performance of existing baselines by employing supported environments that incorporate various VRP variant tasks and instance distributions (termed MTPOMO and MDPOMO, respectively). Empirical results on CVRPLib, shown in Table 4, reveal that training on different tasks significantly enhances generalization performance. This finding underscores the necessity of building foundational models across diverse CO domains.

Large-Scale Instances. We evaluate large-scale CVRP instances of thousands of nodes, with more visualizations and scaling in Appendix. The last row of Table 5 illustrates the performance of the hybrid NAR/AR GLOP [127], while others refer to reproduced results from Ye et al. [127]. Our implementation in RL4CO improves the performance in not only speed but also solution quality.

5.3 Combining Construction and Improvement: Best of Both Worlds?

While constructive policies can build solutions in seconds, their performance is often limited, even with advanced decoding schemes such as sampling or augmentations. On the other hand, improvement methods are more suitable for larger computing budgets. We benchmark models on TSP with 50 nodes: the AR constructive method POMO [62] and the improvement methods DACT [78] and NeuOpt [80]. In the original implementation, DACT and NeuOpt started from a solution constructed randomly. To further demonstrate the flexibility of RL4CO, we show that bootstrapping improvement methods with constructive ones enhance convergence speed. Fig. 5 shows that bootstrapping with a pre-trained POMO policy significantly enhances the convergence speed. To further investigate the performance, we report the Primal Integral (PI) [8, 113, 111], which evaluates the evolution of solution quality over time. Improvement methods alone, such as DACT and NeuOpt, achieve 2.99 and 2.26 respectively, while sampling from POMO achieves 0.08. This shows that the "area under the curve" can be better even if the final solution is worse for constructive methods. Bootstrapping

Benchmark	POMO		MTPOMO		MDPOMO	
	Obj.	Gap	Obj.	Gap	Obj.	Gap
Set A	1075	3.13%	1076	3.20%	**1074**	**2.97%**
Set B	996	3.41%	1003	4.06%	**995**	**3.26%**
Set E	761	5.04%	**760**	**4.82%**	762	5.07%
Set F	813	13.52%	**798**	**12.09%**	825	13.66%
Set M	1259	16.37%	**1234**	**13.58%**	1263	16.03%
Set P	620	6.72%	**608**	**3.72%**	613	5.04%
Set X	73953	16.80%	**73763**	**16.69%**	81848	23.69%

Table 4: Results on CVRPLIB instances with models trained on $N = 50$. Greedy multi-start decoding is used.

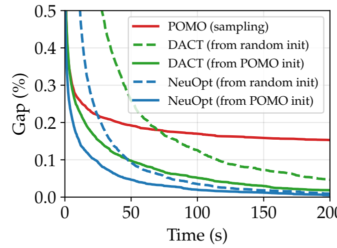

Figure 5: Bootstrapping improvement with constructive methods.

with POMO then improves DACT and NeuOpt to 0.08 and 0.04 respectively, showing the benefits of modularity and hybridization of different components.

6 Discussion

6.1 Limitations and Future Directions

While RL4CO is an efficient and modular library specialized in CO problems, it might not be suitable for any other task due to a number of area-specific optimizations, and we do not expect it to seamlessly integrate with, for instance, OpenAI Gym wrappers without some modifications. Another limitation of the library is its scope so far, namely RL. In fact, extending the library to support supervised methods and creating a comprehensive "AI4CO" library could benefit the whole NCO community. We additionally identify in Foundation Models[3] for CO and related scalable architectures a promising area of future research to overcome generalization issues across tasks and distributions, for which we provided some early clues.

Table 5: Performance on large-scale CVRP instances.

	CVRP1K		CVRP2K		CVRP7K	
	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)
LKH-3	46.4	6.2	64.9	20	245.0	501
AM	61.4	0.6	114.4	1.9	354.3	26
TAM(AM)	50.1	0.8	74.3	2.2	233.4	26
TAM(LKH-3)	46.3	1.8	64.8	5.6	196.9	33
GLOP-G(AM)*	47.1	0.4	63.5	1.2	191.7	2.4
GLOP-G(LKH-3)*	45.9	1.1	63.0	1.5	191.2	5.8
GLOP-G(AM)	46.9	**0.3**	64.7	**0.7**	190.9	**2.0**
GLOP-G(LKH-3)	**45.5**	0.5	**62.8**	0.8	**190.1**	3.9

6.2 Long-term Plans

Our long-term plan is to become the go-to RL for CO benchmark library. While not strictly tied to implementation and benchmarking, we are committed to helping resolve issues and questions from the community. For this purpose, we created a Slack workspace (link available in the online documentation) that by now has attracted more than 130 researchers. It is our hope that our work will ultimately benefit the NCO field with new ideas and collaborations.

7 Conclusion

This paper introduces RL4CO, a modular, flexible, and unified Reinforcement Learning (RL) for Combinatorial Optimization (CO) benchmark. We provide a comprehensive taxonomy from environments to policies and RL algorithms that translate from theory to practice to software level. Our benchmark library aims to fill the gap in unifying implementations in RL for CO by utilizing several best practices with the goal of providing researchers and practitioners with a flexible starting point for NCO research. We provide several experimental results with insights and discussions that can help identify promising research directions. We hope that our open-source library will provide a solid starting point for NCO researchers to explore new avenues and drive advancements. We warmly welcome researchers and practitioners to actively participate and contribute to RL4CO.

[3] https://github.com/ai4co/awesome-fm4co

Acknowledgements

We want to express our gratitude towards anonymous reviewers of previous submissions who greatly helped us improve our paper. Even though rejections were not easy at first, they helped us refine our benchmark. Importantly, through our journey, we got to know several outstanding researchers in the community, who gave us even more motivation and meaning behind our work. We would also like to thank people in the AI4CO open research community who have contributed, and those who will, to RL4CO. We also thank OMELET for supporting us with additional compute. We invite practitioners and researchers to join us and contribute with bug reporting, feature requests, or collaboration ideas. A special thanks also goes to the TorchRL team for helping us in solving issues and improving the library.

Potential Broader Impact

This paper presents work in the field of AI4CO. The main consequene may be that AI methods to solve CO problems may become accessible to the broad public, as our librabry is open source and readily available on GitHub. We do not see potential negative societal consequences as of today.

Funding

This work was supported by a grant of the KAIST-KT joint research project through AI2XL Laboratory, Institute of Convergence Technology, funded by KT [Project No. G01210696, Development of Multi-Agent Reinforcement Learning Algorithm for Efficient Operation of Complex Distributed Systems] and by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government(MSIT)[2022-0-01032, Development of Collective Collaboration Intelligence Framework for Internet of Autonomous Things].

References

[1] A. AhmadiTeshnizi, W. Gao, and M. Udell. OptiMUS: Scalable optimization modeling with (mi) lp solvers and large language models. In *International Conference on Machine Learning*, 2024.

[2] S. Ahn, Y. Seo, and J. Shin. Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*, pages 134–144. PMLR, 2020.

[3] K. Ali, W. Alsalih, and H. Hassanein. Set-cover approximation algorithms for load-aware readers placement in RFID networks. In *2011 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2011.

[4] B. Balaji, J. Bell-Masterson, E. Bilgin, A. Damianou, P. M. Garcia, A. Jain, R. Luo, A. Maggiar, B. Narayanaswamy, and C. Ye. Orl: Reinforcement learning benchmarks for online stochastic optimization problems. *arXiv preprint arXiv:1911.10641*, 2019.

[5] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[6] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning, 2017.

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

[8] T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6): 611–614, 2013.

[9] F. Berto, C. Hua, N. G. Zepeda, A. Hottung, N. Wouda, L. Lan, K. Tierney, and J. Park. RouteFinder: Towards foundation models for vehicle routing problems, 2024. GitHub repository: https://github.com/ai4co/routefinder.

[10] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, et al. The SCIP optimization suite 8.0. *arXiv 2112.08872*, 2021.

[11] J. Bi, Y. Ma, J. Wang, Z. Cao, J. Chen, Y. Sun, and Y. M. Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. *Advances in Neural Information Processing Systems*, 35:31226–31238, 2022.

[12] D. Biagioni, C. E. Tripp, S. Clark, D. Duplyakin, J. Law, and P. C. S. John. graphenv: a python library for reinforcement learning on graph search spaces. *Journal of Open Source Software*, 7(77):4621, 2022.

[13] L. Bodin. Routing and scheduling of vehicles and crews. *Computer & Operations Research*, 10(2):69–211, 1983.

[14] C. Bonnet, D. Luo, D. Byrne, S. Surana, S. Abramowitz, P. Duckworth, V. Coyette, L. I. Midgley, E. Tegegn, T. Kalloniatis, O. Mahjoub, M. Macfarlane, A. P. Smit, N. Grinsztajn, R. Boige, C. N. Waters, M. A. Mimouni, U. A. M. Sob, R. de Kock, S. Singh, D. Furelos-Blanco, V. Le, A. Pretorius, and A. Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2024. URL InternationalConferenceonLearningRepresentations.

[15] A. Bou, M. Bettini, S. Dittert, V. Kumar, S. Sodhani, X. Yang, G. D. Fabritiis, and V. Moens. TorchRL: A data-driven decision-making library for pytorch. In *International conference on learning representations*, 2024.

[16] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[17] P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3):157–183, 1993.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[19] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2019.

[20] F. Bu, H. Jo, S. Y. Lee, S. Ahn, and K. Shin. Tackling prevalent conditions in unsupervised combinatorial optimization: Cardinality, minimum, covering, and more. In *International Conference on Machine Learning*, 2024.

[21] I.-M. Chao, B. L. Golden, and E. A. Wasil. A fast and effective heuristic for the orienteering problem. *European journal of operational research*, 88(3):475–489, 1996.

[22] X. Chen and Y. Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 2019.

[23] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[24] S. Dalton et al. Accelerating reinforcement learning through gpu atari emulation. *Advances in Neural Information Processing Systems*, 33:19773–19782, 2020.

[25] T. Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[26] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359, 2022.

[27] V. C. David Applegate, Robert Bixby and W. Cook. Concorde TSP solver, 2023. URL https://www.math.uwaterloo.ca/tsp/concorde/index.html.

[28] M. Dorigo and T. Stützle. *Ant colony optimization: overview and recent advances*. Springer, 2019.

[29] D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli. Bq-nco: Bisimulation quotienting for generalizable neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 2023.

[30] Z. Drezner and H. W. Hamacher. *Facility location: applications and theory*. Springer Science & Business Media, 2004.

[31] W. Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019. URL https://github.com/Lightning-AI/lightning.

[32] M. Fischetti, J. J. S. Gonzalez, and P. Toth. Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148, 1998.

[33] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax.

[34] B. L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318, 1987.

[35] L. Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL http://www.gurobi.com.

[36] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[37] K. Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12 2017. doi: 10.13140/RG.2.2.25569.40807.

[38] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

[39] A. Hottung, B. Bhandari, and K. Tierney. Learning a latent search space for routing problems using variational autoencoders. In *International Conference on Learning Representations*, 2020.

[40] A. Hottung, Y.-D. Kwon, and K. Tierney. Efficient active search for combinatorial optimization problems. *International conference on learning representations*, 2022.

[41] A. Hottung, M. Mahajan, and K. Tierney. PolyNet: Learning diverse solution strategies for neural combinatorial optimization. *arXiv preprint arXiv:2402.14048*, 2024.

[42] C. D. Hubbs, H. D. Perez, O. Sarwar, N. V. Sahinidis, I. E. Grossmann, and J. M. Wassick. Or-gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319*, 2020.

[43] J. Hwang, J. S. Pak, D. Yoon, H. Lee, J. Jeong, Y. Heo, and I. Kim. Enhancing on-die pdn for optimal use of package pdn with decoupling capacitor. In *2021 IEEE 71st Electronic Components and Technology Conference (ECTC)*, pages 1825–1830, 2021. doi: 10.1109/ECTC32696.2021.00288.

[44] Z. Iklassov, Y. Du, F. Akimov, and M. Takac. Self-guiding exploration for combinatorial problems. *arXiv preprint arXiv:2405.17950*, 2024.

[45] L. Ivan. Capacitated vehicle routing problem library. http://vrp.atd-lab.inf.puc-rio.br/index.php/en/. 2014.

[46] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

[47] Y. Jiang, Y. Wu, Z. Cao, and J. Zhang. Learning to solve routing problems via distributionally robust optimization. In *36th AAAI Conference on Artificial Intelligence*, 2022.

[48] C. K. Joshi, T. Laurent, and X. Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[49] J. Juang, L. Zhang, Z. Kiguradze, B. Pu, S. Jin, and C. Hwang. A modified genetic algorithm for the selection of decoupling capacitors in pdn design. In *2021 IEEE International Joint EMC/SI/PI and EMC Europe Symposium*, pages 712–717, 2021. doi: 10.1109/EMC/SI/PI/EMCEurope52599.2021.9559292.

[50] B. Kalantari, A. V. Hill, and S. R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22(3):377–386, 1985.

[51] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

[52] D. Kikuta, H. Ikeuchi, K. Tajiri, and Y. Nakano. RouteExplainer: An explanation framework for vehicle routing problem. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 30–42. Springer, 2024.

[53] H. Kim, M. Kim, F. Berto, J. Kim, and J. Park. DevFormer: A symmetric transformer for context-aware device placement. *International Conference on Machine Learning*, 2023.

[54] M. Kim, J. Park, and J. Kim. Learning collaborative policies to solve np-hard routing problems. In *Advances in Neural Information Processing Systems*, 2021.

[55] M. Kim, J. Park, and J. Park. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 2022.

[56] M. Kim, S. Choi, J. Son, H. Kim, J. Park, and Y. Bengio. Ant colony sampling with GFlowNets for combinatorial optimization. *arXiv preprint arXiv:2403.07041*, 2024.

[57] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[58] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[59] V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[60] W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *International Conference on Learning Representations*, 2019.

[61] W. Kool, H. Van Hoof, and M. Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, pages 3499–3508. PMLR, 2019.

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

[63] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34: 5138–5149, 2021.

[64] G. Laporte and S. Martello. The selective travelling salesman problem. *Discrete applied mathematics*, 26(2-3):193–207, 1990.

[65] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys. The traveling salesman problem: A guided tour of combinatorial optimization. *The Journal of the Operational Research Society*, 37(5): 535, 1986.

[66] G. Li, C. Xiong, A. Thabet, and B. Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

[67] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, and J. Zhang. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):2306–2315, 2021.

[68] J. Li, Y. Ma, Z. Cao, Y. Wu, W. Song, J. Zhang, and Y. M. Chee. Learning feature embedding refiner for solving vehicle routing problems. *IEEE Transactions on Neural Network and Learning Systems*, 2023.

[69] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[70] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, 85, 2017.

[71] J. T. Linderoth, A. Lodi, et al. Milp software. *Wiley encyclopedia of operations research and management science*, 5:3239–3248, 2010.

[72] F. Liu, X. Lin, Q. Zhang, X. Tong, and M. Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024.

[73] F. Liu, X. Tong, M. Yuan, X. Lin, F. Luo, Z. Wang, Z. Lu, and Q. Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning*, 2024.

[74] R. Lotfi, A. Mostafaeipour, N. Mardani, and S. Mardani. Investigation of wind farm location planning by considering budget constraints. *International Journal of Sustainable Energy*, 37 (8):799–817, 2018.

[75] F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36, 2024.

[76] F. Luo, X. Lin, Z. Wang, T. Xialiang, M. Yuan, and Q. Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024.

[77] L. Luttmann and L. Xie. Neural combinatorial optimization on heterogeneous graphs: An application to the picker routing problem in mixed-shelves warehouses. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 351–359, 2024.

[78] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems*, 34, 2021.

[79] Y. Ma, J. Li, Z. Cao, W. Song, H. Guo, Y. Gong, and Y. M. Chee. Efficient neural neighborhood search for pickup and delivery problems. *arXiv preprint arXiv:2204.11399*, 2022.

[80] Y. Ma, Z. Cao, and Y. M. Chee. Learning to search feasible and infeasible regions of routing problems with flexible neural k-opt. *Advances in Neural Information Processing Systems*, 36, 2024.

[81] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. I: High performance GPU-based physics simulation for robot learning, 2021.

[82] S. Manchanda, S. Michel, D. Drakulic, and J.-M. Andreoli. On the generalization of neural combinatorial optimization heuristics. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part V*, pages 426–442. Springer, 2023.

[83] V. Marianov, D. Serra, et al. Location problems in the public sector. *Facility location: Applications and theory*, 1:119–150, 2002.

[84] Y. Min, Y. Bai, and C. P. Gomes. Unsupervised learning for solving the travelling salesman problem. In *Neural Information Processing Systems*, 2023.

[85] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. *Advances in neural information processing systems*, 27, 2014.

[86] V. Moens. TensorDict: your PyTorch universal data carrier, 2023. URL https://github.com/pytorch-labs/tensordict.

[87] S. A. Mulder and D. C. Wunsch II. Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. *Neural Networks*, 16(5-6): 827–832, 2003.

[88] A. T. Murray, K. Kim, J. W. Davis, R. Machiraju, and R. Parent. Coverage optimization to support security monitoring. *Computers, Environment and Urban Systems*, 31(2):133–147, 2007.

[89] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

[90] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[91] M. Pagliardini, D. Paliotta, M. Jaggi, and F. Fleuret. Faster causal attention over large sequences through sparse flash attention. *arXiv preprint arXiv:2306.01160*, 2023.

[92] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[93] L. Perron and V. Furnon. OR-Tools, 2023. URL https://developers.google.com/optimization/.

[94] A. Prouvost, J. Dumouchelle, L. Scavuzzo, M. Gasse, D. Chételat, and A. Lodi. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. URL https://openreview.net/forum?id=IVc9hqgibyB.

[95] C. PyVRP. PyVRP, 2023. URL https://pyvrp.org/.

[96] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[97] G. K. Rand. Sequencing and scheduling: An introduction to the mathematics of the job-shop. *Journal of the Operational Research Society*, 33:862, 1982. URL https://api.semanticscholar.org/CorpusID:62592932.

[98] B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[99] M. W. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995.

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

[101] W. Shan, Q. Yan, C. Chen, M. Zhang, B. Yao, and X. Fu. Optimization of competitive facility location for chain stores. *Annals of Operations research*, 273:187–205, 2019.

[102] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

[103] J. Son, M. Kim, S. Choi, and J. Park. Solving np-hard min-max routing problems as sequential generation with equity context. *arXiv preprint arXiv:2306.02689*, 2023.

[104] J. Son, M. Kim, H. Kim, and J. Park. Meta-SAGE: Scale meta-learning scheduled adaptation with guided exploration for mitigating scale shift on combinatorial optimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 32194–32210. PMLR, 2023.

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

[106] W. Song, X. Chen, Q. Li, and Z. Cao. Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19(2):1600–1610, 2022.

[107] L. Sun, W. Huang, P. S. Yu, and W. Chen. Multi-round influence maximization. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2249–2258, 2018.

[108] Z. Sun and Y. Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *arXiv preprint arXiv:2302.08224*, 2023.

[109] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[110] P. Tassel, M. Gebser, and K. Schekotihin. A reinforcement learning environment for job-shop scheduling. *arXiv preprint arXiv:2104.03760*, 2021.

[111] D. Thyssens, T. Dernedde, J. K. Falkner, and L. Schmidt-Thieme. Routing arena: A benchmark suite for neural routing solvers. *arXiv preprint arXiv:2310.04140*, 2023.

[112] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[113] T. Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap* neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[114] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28, pages 2692–2700. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

[115] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[116] C. P. Wan, T. Li, and J. M. Wang. RLOR: A flexible framework of deep reinforcement learning for operation research. *arXiv preprint arXiv:2303.13117*, 2023.

[117] R. Wang, L. Shen, Y. Chen, X. Yang, D. Tao, and J. Yan. Towards one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-constrained case. In *The Eleventh International Conference on Learning Representations*, 2022.

[118] S. Wasserkrug, L. Boussioux, D. d. Hertog, F. Mirzazadeh, I. Birbil, J. Kurtz, and D. Maragno. From large language models and optimization to decision optimization CoPilot: A research manifesto. *arXiv preprint arXiv:2402.16269*, 2024.

[119] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022.

[120] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9):5057–5069, 2021.

[121] Z. Xiao, D. Zhang, Y. Wu, L. Xu, Y. J. Wang, X. Han, X. Fu, T. Zhong, J. Zeng, M. Song, and G. Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations*, 2024.

[122] L. Xin, W. Song, Z. Cao, and J. Zhang. Generative adversarial training for neural combinatorial optimization models, 2022. URL https://openreview.net/forum?id=9vsRT9mc7U.

[123] O. Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL https://github.com/facebookresearch/hydra.

[124] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

[125] H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li. Deepaco: Neural-enhanced ant systems for combinatorial optimization. *arXiv preprint arXiv:2309.14032*, 2023.

[126] H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, and G. Song. Large language models as hyper-heuristics for combinatorial optimization. *arXiv preprint arXiv:2402.01145*, 2024.

[127] H. Ye, J. Wang, H. Liang, Z. Cao, Y. Li, and F. Li. GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20284–20292, 2024.

[128] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1621–1632, 2020.

[129] D. Zhang, H. Dai, N. Malkin, A. C. Courville, Y. Bengio, and L. Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 11952–11969. Curran Associates, Inc., 2023.

[130] J. Zhou, Y. Wu, W. Song, Z. Cao, and J. Zhang. Towards omni-generalizable neural methods for vehicle routing problems. In *International Conference on Machine Learning*, 2023.

[131] J. Zhou, Z. Cao, Y. Wu, W. Song, Y. Ma, J. Zhang, and C. Xu. MVMoE: Multi-task vehicle routing solver with mixture-of-experts. In *International Conference on Machine Learning*, 2024.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Each claim has the corresponding contents in the manuscript.
   (b) Did you describe the limitations of your work? [Yes] See § 6.1.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See § 7.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read them and make sure that our paper conform to them.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We do not present theoretical results in this work.
   (b) Did you include complete proofs of all theoretical results? [N/A] We do not present theoretical results in this work.
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We made the whole project open-sourced at `https://github.com/ai4co/rl4co`.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We note that, as common practice in the field, we did not report multiple runs for the main tables as algorithms can take more than one day each to train. However, for experiments limited in the number of samples, such as for the sample efficiency experiments and the mDPP benchmarking, we reported multiple runs with different random seeds, where we demonstrated the robustness of different runs to random seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] All the assets are properly cited.
   (b) Did you mention the license of the assets? [Yes] See Appendix.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] All the new assets are available at `https://github.com/ai4co/rl4co`.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] We discussed the licenses under which we obtained access to the assets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The assets in this work do not involve personally identifiable information or offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] To the best of our knowledge, this work does not involve crowdsourcing or human subjects.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] This work does not involve crowdsourcing or human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] This work does not involve crowdsourcing or human subjects.