Reinforcement Learning-based Heuristics to Guide Domain-Independent Dynamic Programming

Minori Narita[1][0000−0003−2808−6056], Ryo Kuroiwa[2][0000−0002−3753−1644], and J. Christopher Beck[1][0000−0002−4656−8908]

[1] University of Toronto, 5 King's College Road, Toronto, Ontario, Canada
minori.narita@mail.utoronto.ca, jcb@mie.utoronto.ca
[2] National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
kuroiwa@nii.ac.jp

Abstract. Domain-Independent Dynamic Programming (DIDP) is a state-space search paradigm based on dynamic programming for combinatorial optimization. In its current implementation, DIDP guides the search using user-defined dual bounds. Reinforcement learning (RL) is increasingly being applied to combinatorial optimization problems and shares several key structures with DP, being represented by the Bellman equation and state-based transition systems. We propose using reinforcement learning to obtain a heuristic function to guide the search in DIDP. We develop two RL-based guidance approaches: value-based guidance using Deep Q-Networks and policy-based guidance using Proximal Policy Optimization. Our experiments indicate that RL-based guidance significantly outperforms standard DIDP and problem-specific greedy heuristics with the same number of node expansions. Further, despite longer node evaluation times, RL guidance achieves better run-time performance than standard DIDP on three of four benchmark domains.

Keywords: Dynamic Programming · Reinforcement Learning · Deep Learning · Machine Learning · Optimization

1 Introduction

Domain-Independent Dynamic Programming (DIDP) is a state-space search paradigm based on dynamic programming (DP) and heuristic state-space search [17,18]. Previous work has shown DIDP to be competitive with Constraint Programming (CP) and Mixed-Integer Programming (MIP) on a number of benchmark problem classes in combinatorial optimization. Current DIDP solvers guide search with an f-value computed at each state, where $f(s) = g(s) + h(s)$; $g(s)$ is the path cost to the current state s and $h(s)$ is a heuristic that estimates the cost from s to a base state. In its current implementation, DIDP uses user-defined dual bounds as the h-value. However, such dual bounds may not always be very informative and a stronger heuristic guidance could improve solver performance.

Reinforcement learning (RL) has achieved remarkable performance in fields such as control tasks and games [23,24,31,33], and is increasingly being applied to combinatorial optimization [6,8,9,25,38]. The goal of the RL agent is to learn an optimal policy for the given task through trial-and-error interactions with an environment described as a Markov Decision Process (MDP) [2,14]. RL shares several key structures with DP, being represented by the Bellman equation and state-based transition systems. Cappart et al. [8] formulated optimization problems as dynamic programs to bridge an RL model and a CP model, enabling RL-based guidance for variable selection in CP solvers. However, their framework restricts CP formulations to be compatible with the DP model, which can limit performance. In contrast, RL-guided DIDP leverages the alignment between RL and DP models, which may offer a natural integration of RL into exact solvers.

In this paper, we investigate two ways to guide the search in DIDP using RL, value-based and policy-based guidance, and evaluate them on four combinatorial optimization problems. The key contributions of this paper are as follows:

- We introduce two approaches – value-based guidance and policy-based guidance – that effectively direct the search in DIDP;
- We demonstrate that an RL model can be systematically mapped from a DIDP model, establishing a basis for automated mapping in future work;
- Our experimental evaluation shows that DIDP with RL guidance outperforms DIDP based on node expansions and, to a lesser extent, on run-time.

2 Background

In this section, we describe the two foundations of our work: Domain-Independent Dynamic Programming (DIDP) and reinforcement learing (RL).

2.1 DIDP

In DIDP, the user defines a DP model in the Dynamic Programming Description Language (DyPDL) and the model is solved by a solver. While different solving approaches are possible, thus far existing solvers are based on heuristic search.

DyPDL DyPDL is a solver-independent formalism to define a DP model [19] represented as a tuple $\langle \mathcal{V}, s_0, \mathcal{T}, \mathcal{B}, \mathcal{C} \rangle$, where $\mathcal{V} = \{v_1, ..., v_n\}$ is the set of state variables, s_0 is the target state, $\mathcal{T}$ is the set of transitions, $\mathcal{B}$ is the set of *base cases*, and $\mathcal{C}$ is the set of state constraints. Each state variable $v_i \in \mathcal{V}$ is either an element, a set, or a number, and has a domain $\mathcal{D}_i$. A state s is a complete assignment to the state variables, represented by a tuple $\langle d_1, ..., d_n \rangle \in \mathcal{D}$ where $\mathcal{D}$ is the cartesian product of $\mathcal{D}_1$ ... $\mathcal{D}_n$. We denote $s[v_i]$ as the value of the v_i in state s. A target state s_0 is the initial state in the transition system, i.e., the state for which the optimal value is to be computed. State constraints $\mathcal{C}$ are conditions on state variables that must be satisfied by all valid states. A base case $\langle \mathcal{C}_B, \mathsf{cost}_B \rangle \in \mathcal{B}$ is a set of conditions $\mathcal{C}_B$ to terminate the transitions

and the associated cost function cost_B. A state that satisfies $\mathcal{C} \cup \mathcal{C}_B$ is called a base state. A transition $\tau \in \mathcal{T}$ is a 4-tuple $\langle \mathsf{eff}_\tau, \mathsf{cost}_\tau, \mathsf{pre}_\tau, \mathsf{forced}_\tau \rangle$. The effect $\mathsf{eff}_\tau : \mathcal{D}_i \to \mathcal{D}_i$ is a function that maps a value of a state variable v to another value. A state transition returns a successor state $s[\![\tau]\!]$ by applying τ to each state variable in s, i.e., $s[\![\tau]\!][v_i] = \mathsf{eff}_\tau[v_i](s)$, $\forall v_i \in \mathcal{V}$. A numeric cost $\mathsf{cost}_\tau(s)$ is associated with each transition τ from a state s. Preconditions pre_τ are conditions on state variables, and τ is *applicable* in a state s only if all preconditions are satisfied, denoted by $s \models \mathsf{pre}_\tau$. The flag $\mathsf{forced}_\tau \in \{\bot, \top\}$ is a boolean value; if forced transitions are applicable at state s, then the first defined one is executed and all other forced and non-forced transitions are ignored.

Let $x = \{x_1, ..., x_m\}$ be a sequence of transitions for a DyPDL model. Then, x is a *solution* to the model if the sequence starts from s_0 and ends at a base state. For minimization problems, the cost of a solution is $\sum_{i=0}^{m-1} \mathsf{cost}_{x_{i+1}}(s_i) + \min_{\{B | B \in \mathcal{B}; s_m \models \mathcal{C}_B\}} \mathsf{cost}_B(s_m)$, where s_i is the state resulting from applying the first i transitions of the solution from s_0. For maximization, min is replaced with max. We can represent a DyPDL model by a recursive equation called a Bellman equation [4]. The Bellman equation $V(s)$ returns the optimal cost starting from state s where $V(s) = \infty$ (or $V(s) = -\infty$ for maximization) if there does not exist a base state reachable from s. For the minimization (maximization) problem, $\eta(s)$ is a dual bound function iff $\eta(s) \leq V(s)$ ($\eta(s) \geq V(s)$), $\forall s \in \mathcal{D} \wedge s \models \mathcal{C}$.

State-based heuristic search Kuroiwa and Beck [19] implement seven heuristic search algorithms for DIDP based on the literature. Starting from s_0, each algorithm *expands* states, generating a successor state $s' = s[\![\tau]\!]$ for each transition applicable in s. At each successor state, the f-value is computed as $f(s') = g(s') + h(s')$, where $g(s')$ is the path cost from s_0 to s', and $h(s')$ is the heuristic estimate of the cost from s' to a base state. Given a sequence of transitions $x = \{x_1, ..., x_j\}$ to reach a state s' $(= s_j)$ from s_0, the path cost for s' is defined as $g(s') = \sum_{i=0}^{j-1} \mathsf{cost}_{x_{i+1}}(s_i)$. User-defined dual bounds $\eta(s)$ and state constraints $\mathcal{C}$ are used for pruning. Let $\bar{\zeta}$ be the the primal bound. Then, we can prune the node s if $g(s) + \eta(s) \geq \bar{\zeta}$ (for maximization, $g(s) + \eta(s) \leq \bar{\zeta}$). By default, $h(s') = \eta(s')$. For formal details, see Kuroiwa and Beck [19].

In Section 4, we used three search algorithms: complete anytime beam search (CABS), anytime column progressive search (ACPS), and anytime pack progressive search (APPS). Algorithm details are in Appendix A in the arXiv version.[3]

2.2 Reinforcement Learning

Reinforcement learning (RL) [35] is a framework for learning to achieve a goal from interaction with the environment. In RL, the agent operates based on a Markov Decision Process (MDP) [28], defined as a 4-tuple $\langle \mathrm{S}, \mathrm{A}, \mathrm{T}, \mathrm{R} \rangle$, where S is a set of states, A is a set of actions, $\mathrm{T} : \mathrm{S} \times \mathrm{A} \to \mathrm{S}$ is the transition function, and $\mathrm{R} : \mathrm{S} \times \mathrm{A} \to \mathbb{R}$ is the reward function. For simplicity, we use the notation s

[3] `https://arxiv.org/abs/2503.16371`

to represent a state in the MDP as well as a state in the DP. The initial state s_0 is sampled from an initial state distribution ρ_0: $\mathbb{R} \to \mathrm{S}$. At each time-step, the agent performs an action $a \in \mathrm{A}$ at current state s, which brings the agent to the next state s' and gives a reward $r = \mathrm{R}(s, a)$. We assume a deterministic transition function, so $\mathrm{T}(s, a)$ returns the next state s'. An episode terminates when the agent reaches a terminal state. The goal of the RL agent is to learn an optimal policy π, that maximizes the expected total reward $\sum_{t=0}^{\infty} \gamma^t r_t$, given the initial state distribution ρ_0, where γ is a discount factor ($0 \leq \gamma \leq 1$) and t is a time-step. A policy $\pi : \mathrm{S} \times \mathrm{A} \to [0, 1]$ is a conditional distribution over actions given the state, indicating the likelihood of the agent choosing an action. Policy-based methods like TRPO [30] and PPO [31] directly optimize the policy through policy gradient methods. Value-based methods such as DQN [24] learn a value function from exploration. For instance, DQN learns an estimated Q-value function $Q^{\pi}(s, a)$, the expected return for selecting action a at state s if the agent follows policy π afterwards.

RL is increasingly being applied to solve combinatorial optimization problems. Early approaches used recurrent neural networks to learn constructive heuristics for generating solutions [5]. However, graph neural networks (GNNs) have become more prevalent [9,16,25] as they are size-agnostic and permutation-invariant. A limitation of end-to-end RL methods is the challenge of managing constraints, as well as the lack of a systematic way to improve the obtained solutions, unlike exact methods such as CP and MIP [7,8].

Significant progress has recently been made in combining search with learned heuristics to address these limitations [8,11,15,34]. Kool et al. [15] formulated routing problems as a DP problem and guided the search using a learned heat map of edge weights. Cappart et al. [8] formulated optimization problems as DP models to bridge an RL model and a CP model, enabling RL-based guidance for variable selection in CP solvers.

Guiding search using deep reinforcement learning has also been explored in AI planning. Orseau et al. [26,27] proposed learning Q-value functions and policies to weight nodes in best-first search to solve two-dimensional single-agent problems like Sokoban. DeepCubeA [1] tackled the Rubik's Cube by learning a heuristic function through approximated value iteration and applying it in batch-weighted A* search. Lastly, Gehring et al. [12] leveraged domain-independent heuristic functions as dense reward generators to train RL agents, and used the RL-based heuristics to improve search efficiency for classical planning problems.

3 RL-based search guidance for DIDP

Given the similarities in the state-based formulations of DIDP and RL, it is natural to investigate the guidance of search in DIDP based on a heuristic function learned with RL. Each component of an MDP can be systematically derived from the corresponding component in a DIDP model. For instance, the set of states in the DIDP model matches precisely with the state space in the MDP and transitions in the DIDP model have a 1-to-1 mapping to actions in the MDP.

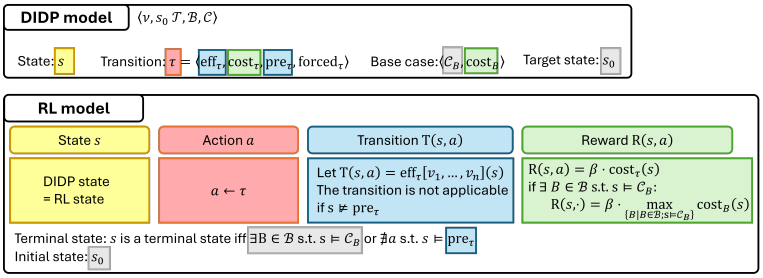

Fig. 1. Mapping from a DIDP model to an RL model for maximization problems. State constraints $\mathcal{C}$ and forced_τ are not mapped to the RL model.

An RL agent trained on the mapped MDP can then serve as a heuristic function for computing f-values in the DIDP search.

In this paper, we develop a scheme to convert a DIDP model to an RL model for a given combinatorial optimization problem. Figure 1 provides an overview of how an RL model is mapped from a DIDP model. Each component in the MDP is derived from the DIDP model as follows:

- `State` $s \in \mathrm{S}$: The same state space as in the DIDP model.
- `Action` $a \in \mathrm{A}$: 1-to-1 mapping from transition $\tau \in \mathcal{T}$, i.e., $\tau \mapsto a$.
- `Transition function` $\mathrm{T}(s,a)$: mapped from pre_τ and eff_τ in each $\tau \in \mathcal{T}$. For a state s, the next state reached by taking an action a is obtained by applying eff_τ to each state variable in s, i.e., $\mathrm{T}(s,a) = \mathsf{eff}_\tau[v_1, ..., v_n](s)$. The transition is not applicable if the preconditions are not met, i.e., $s \not\models \mathsf{pre}_\tau$. Thus, the mapping includes the masking of non-applicable actions as is common in RL models with invalid actions [8,13,33].
- `Reward function` $\mathrm{R}(s,a)$ corresponds to the transition cost incurred by applying τ at state s, $\mathsf{cost}_\tau(s)$. To improve the stability of RL training, the reward is scaled by a hyperparameter β. For a minimization problem, the reward has to be negated to make the RL task a maximization problem, i.e., $\mathrm{R}(s,a) = -\beta \cdot \mathsf{cost}_\tau(s)$. If s is the base state, cost_B is applied instead.

A state s is a terminal state if s satisfies at least one base case, i.e., $\exists \mathrm{B} \in \mathcal{B}$ s.t. $s \models \mathcal{C}_\mathrm{B}$, or there is no action that satisfies preconditions, i.e., $\nexists\, a$ s.t. $s \models \mathsf{pre}_\tau$ where a is mapped from τ. The initial state in RL is the target state in the DIDP model s_0. The state constraints $\mathcal{C}$ are not mapped to the RL model.[4]

Including domain knowledge and introducing auxiliary rewards are often crucial for successful RL training [21,32]. In this paper, we focus on leveraging the structural relationships between DIDP and RL models to systematically build RL models for search guidance. Future work will examine automating this mapping and incorporating problem-specific structures into the mapping process.

[4] While state constraints $\mathcal{C}$ can be mapped to action masking, the implications of action masking in RL remain underexplored [13]. Thus, the investigation of integrating $\mathcal{C}$ into the MDP is left for future work.

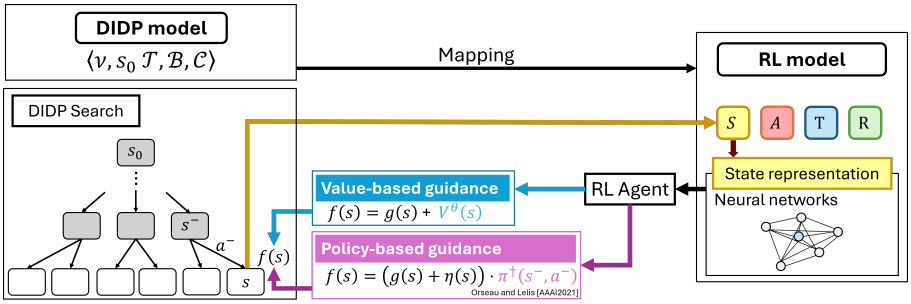

Fig. 2. Value-based guidance and policy-based guidance for DIDP. The equations for computing f-values in the figure are for maximization problems.

3.1 Value-based guidance

The value-based guidance approach directly uses the value function approximated by neural network parameters θ, $V^\theta(s)$, as a heuristic function, where $h(s) = V^\theta(s)$. $V^\theta(s)$ is an estimated total reward from s to a terminal state, which aligns with the definition of $h(s)$, the estimated total cost from s to a base state. For minimization problems, the value function is negated, i.e., $h(s) = -V^\theta(s)$. Figure 2 shows the overview of this approach. First, the MDP is derived from the DIDP model, as described above. Then, the RL agent is trained in the mapped MDP with a value-based RL algorithm (e.g., DQN). During search, every time a node is expanded, the f-value is calculated for all its successor states s by $f(s) = g(s) + V^\theta(s)$; the f-values are then used to determine the priority of the state in the DIDP framework.

Details of the computation for each successor node s are as follows. First, path cost $g(s)$ is calculated by $g(s) = g(s^-)+\mathsf{cost}_\tau(s^-)$, where s^- is the parent node of s and τ is a transition that transforms s^- to s. To align the scale of $g(s)$ with that of $V^\theta(s)$, the scaling factor β is used; thus, $g(s) = g(s^-) + \beta \cdot \mathsf{cost}_\tau(s^-)$. The heuristic value is computed by calling a neural network prediction, i.e., $V^\theta(s) = \mathcal{F}(s;\theta)$. We use DQN as a value-based RL algorithm, so the output is the Q-values for each action a for the input state s. $V^\theta(s)$ is obtained by $V^\theta(s) = \max_{a \in A'} Q^\theta(s,a)$, where A' is the set of all applicable actions in s.

3.2 Policy-based guidance

The policy-based guidance approach uses the same MDP as value-based guidance, but is trained with a policy-based RL algorithm, such as PPO [31]. The algorithm learns a policy $\pi(s,a)$, a probability distribution over actions for a given state. $\pi(s,a)$ is then used to weight the original f-value to prioritize the expansion of the nodes that are deemed promising by the policy.

Details of the computation at each successor node generation are as follows. First, path cost $g(s)$ is calculated by $g(s) = g(s^-)+\mathsf{cost}_\tau(s^-)$. Unlike in

value-based guidance, there is no need to scale the path cost $g(s)$, as the policy has a fixed scale of $[0, 1]$ and is only used to weight the original f-values. Then, the policy $\pi(s^-, a^-)$ is obtained by calling the neural network and obtain the a^-th output, i.e., $\pi(s^-, a^-) = \mathcal{F}(s^-; \theta)[a^-]$.[5] Our approach uses the accumulated probabilities up to state s from the root node, i.e., $\pi^\dagger(s^-, a^-) = \pi(s_0, a_0)\pi(s_1, a_1)...\pi(s^-, a^-)$, to take all the previous decisions up to s into consideration [27]. Therefore, the f-value is computed as $f(s) = (g(s) + \eta(s)) \cdot \pi^\dagger(s^-, a^-)$, where $\eta(s)$ is the dual bound at s. A promising action with a high probability in the policy will have a higher f-value (for maximization), thereby making the corresponding state more likely to be expanded next. For minimization problems, we divide $g(s) + \eta(s)$ by $\pi^\dagger(s^-, a^-)$ instead, i.e., $f(s) = (g(s) + \eta(s))/\pi^\dagger(s^-, a^-)$, so that promising actions yield lower f-values.

3.3 State Representation

The state representation needs to be able to handle instances of different sizes and to be invariant to input permutations [8]. Hence, we used a graph attention network (GAT) [36] as a state representation for routing problems to leverage the natural graph structure of these domains, and a set transformer [20] or Deep Sets [37] for packing problems. The details of the neural network architecture for each problem used in this paper appear in Appendix B in the arXiv version. The embedding obtained by the neural network can then be used as an input to a fully-connected network. For DQN, the network outputs Q-values for each action a for state s, so the output layer is of size $|A|$. For PPO, two separate networks for the actor and the critic are used. The critic network outputs a single value, representing the estimated value of the state, while the actor network applies a softmax activation function to its final layer to output action probabilities $\pi(s, \cdot)$. The outputs are processed as described in Sections 3.1 and 3.2.

4 Experiments

We evaluated our methods on four combinatorial optimization problems: Traveling Salesperson Problem (TSP), TSP with Time Windows (TSPTW), 0-1 Knapsack, and Portfolio Optimization.[6] To assess the quality of RL guidance, the solution quality per node expansion was evaluated for different guidance methods. Our two RL-based guidance methods (h=DQN and π=PPO) were compared with dual-bound guidance (default DIDP implementation), uniform cost search (i.e., $h = 0$ for all states), and greedy heuristic guidance.

The greedy heuristic-based h-value at a state is equal to the cost of the path to a base case found by rolling-out the greedy heuristic from that state. Greedy heuristics exploit domain-specific knowledge from outside the DP model, and thus serve as a baseline to evaluate how RL guidance competes with hand-crafted

[5] In our implementation, the neural network is called only once when s^- is expanded. Each element of $\pi(s^-)$ is assigned to the corresponding successor through indexing.

[6] All code are available at `https://github.com/minori5214/rl-guided-didp`.

heuristics. The definitions of the greedy heuristics for each problem domain and further details are in Appendix C in the arXiv version.

We also evaluated the solution quality after a one hour run-time to compare the performance of our approach with baseline methods. The results include performance from MIP (Gurobi), pure CP (CPLEX CP Optimizer), RL-guided CP (BaB-DQN and RBS-PPO [8]), and sampling-based heuristic methods (dual-bound, greedy, DQN, PPO). Sampling is done by applying a softmax function to the h-values of all successor states to get action probabilities and choosing the next state probabilistically. The number of samples is set to 1280, following Kool et al. [16]. The memory limit for each approach is set to 8 GB.

Evaluation metric: At a given node expansion limit l, the gap is calculated by $gap = |x(i, m, l) - best(i)|/best(i) \times 100$, where $x(i, m, l)$ is the solution cost of method m for instance i up to the node expansion limit l, and $best(i)$ is the best known solution for i. The best known solution includes results from all the approaches, including baselines, within the time limit. If no feasible solution is found within the given node expansion limit, a fixed value of 100 [%] is used.

Training Process: Although the network architectures are size-agnostic, we trained DQN and PPO for each problem size and domain, as examining the scalability of neural networks is not our main focus. Training begins with randomly generating an instance from a fixed distribution using an instance generator. The agent then explores the instance following the current policy π until the agent reaches the terminal state with the experiences stored in the replay buffer. The network parameters θ are updated using the experiences sampled from the replay buffer. The training time is limited to 72 hours. Details of the network architectures and hyperparameters are in Appendix B in the arXiv version.

4.1 Problem Domains

To evaluate our approach, we chose routing problems (TSP and TSPTW) and packing problems (0-1 Knapsack and Portfolio Optimization).

TSP In the Traveling Salesperson Problem (TSP) [10], a set of customers $N = \{0, ..., n\}$ is given, and a solution is a tour starting from the depot ($i = 0$) and returning to the depot, visiting each customer exactly once. Visiting customer j from i incurs the travel time $c_{ij} \geq 0$. TSP instances are generated by removing time window constraints from the TSPTW instances used below.

DIDP model: For TSP, a state is a tuple of variables $\langle U, i \rangle$ where U is the set of unvisited customers and i is the current location. In this model, one customer is visited at each transition. The minimum possible travel time to visit customer j is $c_j^{\text{in}} = \min_{k \in N \setminus \{j\}} c_{kj}$, and the minimum travel time from j is $c_j^{\text{out}} = \min_{k \in N \setminus \{j\}} c_{jk}$. The DIDP model is represented by the following Bell-

man equation, adapted from the TSPTW model defined below.

$$\text{compute } V(N \setminus \{0\}, 0) \tag{1}$$

$$V(U,i) = \begin{cases} c_{i0} & \text{if } U = \emptyset \\ \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j) & \text{if } U \neq \emptyset \end{cases} \tag{2}$$

$$V(U,i) \geq \max \left\{ \sum\nolimits_{j \in U \cup \{0\}} c_j^{\text{in}}, \sum\nolimits_{j \in U \cup \{i\}} c_j^{\text{out}} \right\} \tag{3}$$

Expression (1) declares that the optimal cost is the cost to visit all customers ($U = N \setminus \{0\}$) starting from the depot ($i = 0$). The second line of Eq. (2) corresponds to visiting customer j from i; then, j is removed from U and the current location i is updated to j. The first line of Eq. (2) is the base case, where all customers are visited ($U = \emptyset$) and the recursion ends. Eq. (3) represents two dual bounds.

RL model: The MDP for this DIDP model is defined as follows:

`State` s: $\langle U, i \rangle$
`Action` $a = j \in U$
`Transition function` $\mathrm{T}(s,a)$: $\mathrm{T}(\langle U, i \rangle, j) = \langle U \setminus j, j \rangle$
`Reward function` $\mathrm{R}(s,a)$: $\mathrm{R}(\langle U, i \rangle, j) = \beta \cdot (-c_{ij})$

The reward function is the negative distance between the current location i and the next customer j. The scaling factor is $\beta = 0.001$. Dual bounds are not used in the MDP.

TSPTW In TSP with Time Windows (TSPTW) [29], the visit to customer i must be within a time window $[a_i, b_i]$. If customer i is visited before a_i, the salesperson has to wait until a_i. The instances were generated in the same way as Cappart et al. [8], but the maximal time window length allowed is set $W = 100$, and the maximal gap between two consecutive time windows is set $G = 1000$ to make the instances more challenging.

DIDP model: A state is a tuple $\langle U, i, t \rangle$ where t is the current time. The set of customers that can be visited next is $U' = \{j \in U \mid t + c_{ij} \leq b_j\}$. The DIDP model is represented by the following Bellman equation [19]:

$$\text{compute } V(N \setminus \{0\}, 0, 0) \tag{4}$$

$$V(U,i,t) = \begin{cases} c_{i0} & \text{if } U = \emptyset \\ \min_{j \in U'} c_{ij} + V(U \setminus \{j\}, j, \max(t + c_{ij}, a_j)) & \text{if } U \neq \emptyset \end{cases} \tag{5}$$

$$V(U,i,t) = \infty \qquad \text{if } \exists j \in U, t + c_{ij}^* > b_j \tag{6}$$

$$V(U,i,t) \leq V(U,i,t') \qquad \text{if } t \leq t' \tag{7}$$

$$V(U,i,t) \geq \max \left\{ \sum\nolimits_{j \in U \cup \{0\}} c_j^{\text{in}}, \sum\nolimits_{j \in U \cup \{i\}} c_j^{\text{out}} \right\} \tag{8}$$

In the second line of Eq. (5), time t is updated to $\max(t + c_{ij}, a_j)$. Eq. (6) is a state constraint that sets the value of a state to be infinity if there exists a

customer j that cannot be visited by the deadline b_j even if we take the shortest path with distance c^*_{ij}. Inequality (7) is a dominance relationship; if other state variables are the same in two states, then a state having smaller t always leads to a better solution. Eq. (8) represents two dual bounds.

RL model: The MDP for this DIDP model is defined as follows:

`State` s: $\langle U, i, t \rangle$
`Action` $a = j \in U$
`Transition function` $\mathrm{T}(s, a)$: $\mathrm{T}(\langle U, i, t \rangle, j) = \langle U \setminus j, j, \max(t + c_{ij}, a_j) \rangle$
`Reward function` $\mathrm{R}(s, a)$: $\mathrm{R}(\langle U, i, t \rangle, j) = \beta \cdot (|\mathrm{UB}_{cost}| - c_{ij})$

UB_{cost} is a strict upper bound on the reward of any solution for this problem domain to ensure that the RL agent has the incentive to find feasible solutions first and then to find the best ones. UB_{cost} is not included in the mapping in Figure 1, but this reward structure is originally introduced in Cappart et al. [8] and helps improve the RL training. The scaling factor is set to $\beta = 0.001$.

0-1 Knapsack In the 0-1 Knapsack Problem [22], a set of items $N = \{0, ..., n-1\}$ with weights w_i and profits p_i for $i \in N$ and a knapsack with budget B are given. The objective is to maximize the total profit of the items in the knapsack. The items are sorted in descending order of the profit ratio (p_i/w_i). The instance distribution is taken from the "Hard" instances in Cappart et al. [8], where profits and weights are strongly correlated.

DIDP model: A DIDP state is a tuple $\langle x, i \rangle$, where x is the current total weight and i represents the current item index. The DIDP model is based on Kuroiwa and Beck [19] with the remaining budget replaced by the current total weight x:

$$\text{compute } V(0, 0) \tag{9}$$

$$V(x, i) = \begin{cases} \max\{p_i + V(x + w_i, i + 1), V(x, i + 1)\} & \text{if } i < n \wedge x + w_i \leq B \\ V(x, i + 1) & \text{if } i < n \wedge x + w_i > B \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

$$V(x, i) \leq \min \left\{ \sum_{j=i}^{n-1} p_j, \max_{j \in \{i..n-1\}} \left(\frac{p_j}{w_j} \right) \cdot (B - x) \right\}. \tag{11}$$

Expression (9) declares that the optimal cost is the cost to consider all items starting from the first item ($i = 0$) with the current total weight $x = 0$. The first line of Eq. (10) corresponds to considering item i; if i is taken (the first term), then x is updated to $x + w_i$ and the item index is updated to $i + 1$. If i is not taken (the second term), x remains the same. The second line of Eq. (10) indicates that i cannot be taken if doing so exceeds budget B. The third line is the base case; when all items are visited ($i \geq n$), then the recursion terminates. Eq. (11) represents two dual bounds.

RL model: The MDP for the RL agent is as follows:

`State` s: $\langle x, i \rangle$
`Action` $a \in \{0, 1\}$: whether to take the item i or not.
`Transition function` $\mathrm{T}(s, a)$: $\mathrm{T}(\langle x, i \rangle, a) = \langle aw_i + x, i + 1 \rangle$
`Reward function` $\mathrm{R}(s, a)$: $\mathrm{R}(\langle x, i \rangle, a) = \beta(ap_i)$

The RL state is same as the DP state, and the action set is binary: 0 indicates the item is not selected, while 1 means it is selected. The transition function matches the effect of a transition in the DIDP model. The state variable x is updated to $x + aw_i$, i.e., w_i is added to x if the item is selected ($a = 1$). Also, the item index i is incremented by 1. The reward function corresponds to p_i, the profit of item i. The scaling factor is set to $\beta = 0.0001$.

Portfolio Optimization In the 4-moments portfolio optimization problem [3], a set of investments $N = \{0, ..., n-1\}$, each with a specific cost (w_i), expected return (μ_i), standard deviation (σ_i), skewness (γ_i), and kurtosis (κ_i), and the budget B are given. The goal is to find a portfolio with a maximum return as specified by the objective function (Eq. (12)). Each financial characteristic is weighted (λ_1, λ_2, λ_3, and λ_4). The instance distribution is taken from Cappart et al. [8].
DIDP model: A state is a tuple $\langle x, i, Y \rangle$, where x is the current total weight, i is the current item index, and Y is a set of investments up to i, $\{0, ..., i-1\}$. The objective function value up to item i is defined as follows:

$$\nu(Y) = \lambda_1 \sum_{j \in Y} \mu_j - \lambda_2 \sqrt[2]{\sum_{j \in Y} \sigma_j^2} + \lambda_3 \sqrt[3]{\sum_{j \in Y} \gamma_h^3} - \lambda_4 \sqrt[4]{\sum_{j \in Y} \kappa_j^4}. \tag{12}$$

The transition cost is the difference between the objective value of the current state ($\nu(Y)$) and that of the successor state, i.e., $\nu(Y \cup \{i+1\}) - \nu(Y)$. The DIDP model is expressed as follows:

$$\text{compute } V(0, 0, \emptyset) \tag{13}$$

$$V(x, i, Y) = \begin{cases} \max(\nu(Y \cup \{i\}) - \nu(Y) + V(x + w_i, i + 1, Y \cup \{i\}), \\ \qquad V(x, i + 1, Y)) & \text{if } i < n \wedge x + w_i \leq B \\ V(x, i + 1, Y) & \text{if } i < n \wedge x + w_i > B \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

$$V(x, i, Y) \leq \min \left(\lambda_1 \sum_{j=i}^{n-1} \mu_j + \lambda_3 \sqrt[3]{\sum_{j=i}^{n-1} \gamma_j^3}, \max_{j \in \overline{Y}} K_j \cdot (B - x) \right) \tag{15}$$

where $K_j = \left(\frac{\lambda_1 \mu_j + \lambda_3 \sqrt[3]{\gamma_j^3}}{w_j} \right)$ and $\overline{Y} = \{i..n-1\}$. In the first line of Eq. (14), if item i is taken, Y is updated to $Y \cup \{i\}$. The second and third lines of Eq. (14) are the same as Eq. (10) except that a state includes Y as well. Eq. (15) are two dual bounds. Proofs for the dual bounds are in Appendix D in the arXiv version.

RL model: The MDP for the RL agent is as follows:

`State` s: $\langle x, i, Y \rangle$
`Action`: $a \in \{0, 1\}$
`Transition function` $\mathrm{T}(s, a)$:
$T(\langle x, i, Y \rangle, 1) = \langle w_i + x, i + 1, Y \cup \{i\} \rangle$
$T(\langle x, i, Y \rangle, 0) = \langle x, i + 1, Y \rangle$
`Reward function` $\mathrm{R}(s, a)$: $R(\langle x, i, Y \rangle, a) = a \cdot (\nu(Y \cup \{i\}) - \nu(Y))$

In the transition function, Y is updated to $Y \cup \{i\}$ if i is taken. The scaling factor is set to $\beta = 0.0001$.

5 Results

Figure 3 shows the solution quality per node expansion with different heuristic guidance for each problem and DIDP search algorithm.

TSP The plots highlight the strong performance of PPO guidance (red) compared to other heuristics, including dual-bound (blue) and greedy heuristic (green), across all three solvers. DQN also outperforms the default dual-bound guidance except for APPS, though it falls short of the problem-specific greedy heuristic. Greedy heuristic guidance significantly outperformed dual-bound guidance.

TSPTW The solid yellow and red lines (denoted as "RL=tsptw") represent the performance of DIDP guided by the RL agent trained in the TSPTW environment. The performance of these DIDP was significantly worse than that of dual-bound and greedy heuristic guidance. In fact, their performance was even worse than $h = 0$. Given these results, we experimented with using the TSP RL model to guide the search (dotted lines), which significantly outperformed DIDP guided by the TSPTW RL model but achieved about the same performance as other heuristic guidance methods, including $h = 0$.

0-1 Knapsack All the heuristic guidance quickly achieved solutions with gaps of less than 1%, although dual-bound guidance (blue) exhibited slightly worse performance compared to the others. During training, the policies generated by DQN and PPO rapidly converged to the best-ratio heuristic (greedy heuristic), which explains the similar behavior across these three guidance methods.

Portfolio Optimization The plots highlight that PPO guidance (red) significantly outperforms other guidance, including the problem-specific greedy guidance (green). DQN guidance (yellow) also surpassed the dual-bound guidance (blue) and was competitive with the greedy heuristic.

Table 1 shows the performance of the different methods. DIDP performed best in TSPTW, MIP (Gurobi) in TSP and Knapsack, and CP Optimizer for Portfolio. DIDP guided by PPO outperforms the dual-bound guidance using the same solver in terms of average gap across all problem domains except for TSP with $n = 50$. As CABS with dual-bound guidance is the best performing solver [19], these results suggest that PPO guidance has surpassed the current

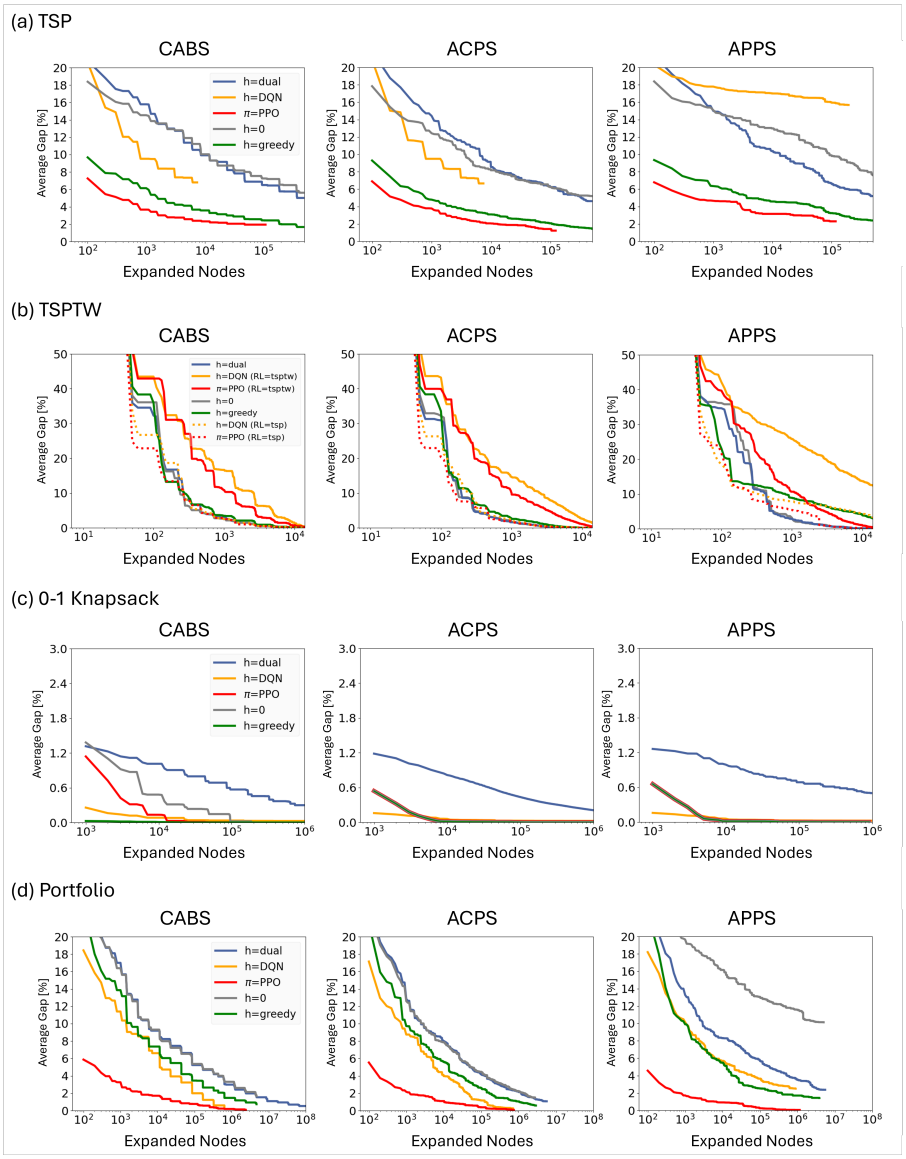

Fig. 3. Results of applying heuristics to guide DIDP, averaged over 40 instances (20 each for small and medium sizes). Small instances have $n = 20$ and medium instances have $n = 50$, except for 0-1 Knapsack ($n = 50$ small, $n = 100$ medium).

state-of-the-art for DIDP in these problems. DIDP guided by DQN also outperformed dual-bound in several settings, though it falls short in TSP.

RL guidance takes orders of magnitude more time for per node expansion due to the call to the neural network prediction. Despite this bottleneck, PPO guidance achieves better performance than the baselines at the time limit. Compared to Cappart et al. [8], the DQN-guided approach in our framework achieves significantly higher performance. For instance, in Portfolio, BaB-DQN achieves an average gap of 10.8%, while CABS (h=DQN) achieves a substantially lower gap of 0.77%. Similarly, DIDP with h=DQN outperforms BaB-DQN in TSPTW, likely because the base DIDP model substantially outperforms the base CP model. PPO guidance exhibits a similar trend, showing notable improvements over RBS-PPO in TSPTW and slightly better results in Portfolio (e.g., 0.50% for RBS-PPO compared to 0.19% for CABS (π=PPO)). However, in TSP, RBS-PPO shows slightly better performance than DIDP guided by PPO.

Table 1 also compares the performance of heuristic sampling against baseline methods. In TSP, PPO clearly outperformed other heuristics, achieving average gaps of 0.26% for $n = 20$ and 3.85% for $n = 50$. In TSPTW, DQN and PPO heuristics were relatively effective in finding feasible solutions (e.g., achieving feasibility in 16 out of 20 instances for $n = 20$), but their ability in optimizing the solution cost is poor (average gaps of 35.12% for DQN and 25.64% for PPO for $n = 20$). For Knapsack, the performance across heuristics was similar, although the dual-bound heuristic was slightly worse. In Portfolio, PPO showed strong standalone performance, being only 0.75% worse than the best-known solution for $n = 50$. The average gap for DQN (3.63% for $n = 20$ and 8.09% for $n = 50$) is comparable to that of the greedy heuristic (5.22% for $n = 20$ and 6.84% for $n = 50$), while the dual-bound heuristic performed considerably worse (4.19% for $n = 20$ and 18.46% for $n = 50$).

6 Discussion

When is RL guidance helpful, and to what extent? RL guidance is most impactful when heuristic quality plays a critical role in solution quality. For instance, in TSPTW, the DIDP model prunes many states by time windows. In such cases, the primary role of heuristic guidance in DIDP appears to be minimizing solution costs rather than ensuring feasibility. In contrast, TSP and Portfolio DP models lack such pruning mechanisms, making heuristic quality a more critical factor in improving the solution quality.

When the performance of DIDP appears to depend primarily on heuristic guidance, the effectiveness of guidance aligns with the performance of the heuristic in sampling-based approaches. For example, PPO guidance consistently outperforms dual-bound guidance because the PPO heuristic is better at driving the search towards high quality solution, as shown in Table 1. Dual-bounds are admissible and thus effective at de-prioritizing unpromising decisions, but may not necessarily guide the search towards more promising solutions.

Solution Quality in Terms of Solve Time While DQN and PPO guidance demonstrate significantly higher solution quality per node expansion compared to dual-bound guidance, their performance gains over time are relatively limited.

Table 1. Comparison of results with baseline methods. Values represent averages over 20 instances. The lowest average gap for each problem and size n is underlined. The symbol * indicates that optimality was proven for all 20 instances. In the DIDP results, values are highlighted in bold if the corresponding method achieves a better average gap than dual-bound guidance using the same solver. For TSP with $n = 50$, sampling with DQN timed-out before completing 1280 samples. "-" denotes reaching either time or memory limits. "t.o." indicates that all 20 instances reached the time limit.

			TSP				TSPTW						0-1 Knapsack				Portfolio			
			n=20		n=50		n=20			n=50			n-50		n=100		n=20		n=50	
Type	Name		Gap	Time	Gap	Time	Feas.	Gap	Time	Feas.	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
CP	CP Optimizer		0.00	25	2.75	t.o.	20	12.04	t.o.	20	32.32	t.o.	0.00	t.o.	0.02	t.o.	0.00*	<1	0.00	t.o.
	BaB-DQN		3.77	t.o.	18.46	t.o.	20	0.00*	216	20	40.83	t.o.	0.00	t.o.	0.00	t.o.	0.00*	1270	10.18	t.o.
	RBS-PPO		0.12	t.o.	1.17	t.o.	20	0.65	t.o.	20	33.82	t.o.	0.00	t.o.	0.00	t.o.	0.00*	487	0.50	t.o.
MIP	Gurobi		0.00*	3	0.03	t.o.	20	0.00*	<1	20	0.00*	119	0.00*	<1	0.00*	<1	-	-	-	-
Heuristics	DQN		2.29	1251	(20.14)	(18239)	16	35.12	1405	0	-	-	0.02	44	0.04	121	3.63	36	8.09	132
	PPO		0.26	249	3.85	1783	20	25.64	423	20	46.93	2050	0.04	51	0.13	132	3.35	38	0.75	111
	Dual-bounds		2.91	4	9.58	1424	0	-	-	0	-	-	0.07	8	0.37	44	4.19	1	18.46	4
	Greedy		5.30	11	8.48	176	0	-	-	0	-	-	0.03	3	0.01	8	5.22	2	6.84	7
DIDP	CABS	h=greedy	0.00	292	2.79	-	20	0.00*	<1	20	0.00*	22	0.00	-	**0.00**	-	0.00*	15	1.48	-
		h=dual	0.00*	19	1.86	-	20	0.00*	<1	20	0.00*	<1	0.00	-	0.09	-	0.00*	2	0.81	-
		h=DQN	0.00	154	12.05	t.o.	20	0.00*	44	20	0.00*	765	0.00	t.o.	**0.05**	t.o.	0.00*	780	**0.77**	t.o.
		π=PPO	0.00	32	3.89	t.o.	20	0.00*	40	20	0.00*	860	0.00	t.o.	**0.00**	t.o.	0.00*	477	**0.19**	t.o.
	ACPS	h=greedy	0.00*	62	**2.45**	-	20	0.00*	<1	20	0.00*	2	0.00	-	**0.00**	-	0.00*	4	**1.17**	-
		h=dual	0.00*	8	6.35	-	20	0.00*	<1	20	0.00*	<1	0.00	-	0.21	-	0.00*	<1	2.14	-
		h=DQN	0.00	371	10.09	t.o.	20	0.00*	11	20	0.00*	99	0.00	t.o.	**0.03**	t.o.	0.00*	180	**0.50**	t.o.
		π=PPO	0.00	345	**2.45**	t.o.	20	0.00*	11	20	0.00*	123	0.00	t.o.	**0.00**	t.o.	0.00*	119	**0.14**	t.o.
	APPS	h=greedy	0.00*	66	**3.46**	-	20	0.00*	<1	20	0.00*	4	0.00	-	**0.00**	-	0.00*	4	**2.86**	-
		h=dual	0.00*	8	8.30	-	20	0.00*	<1	20	0.00*	<1	0.00	-	0.58	-	0.00*	<1	4.74	-
		h=DQN	1.08	t.o.	31.57	t.o.	20	0.00*	13	20	0.00*	141	0.00	t.o.	**0.03**	t.o.	0.00*	187	5.01	t.o.
		π=PPO	0.20	t.o.	**4.17**	t.o.	20	0.00*	12	20	0.00*	141	0.00	t.o.	**0.00**	t.o.	0.00*	119	**0.10**	t.o.

The primary cause lies in the time required to expand a single node. As shown in Table 1, generating a solution using DQN or PPO takes much longer than using dual-bound or greedy heuristics (e.g., DQN takes 313 times longer than dual-bound to sample 1280 times for TSP n=50). While our experiments highlight the potential of RL-based heuristics, they also emphasize the need to address the computational overhead associated with these methods.

7 Conclusion

The initial demonstration of DIDP solver performance was based on search guidance with dual bounds defined in the model. Through experiments on three anytime algorithms, we demonstrated that RL can provide heuristic guidance that improves solution quality with fewer node expansions. These findings show the effectiveness of RL-guided search within anytime algorithms and help to elucidate the conditions where RL guidance is most beneficial, such as in domains where heuristic quality plays a critical role in solution improvement. The inherent structural similarity between DP and RL models offers a natural synergy, enabling RL to be easily integrated into the DIDP framework. With further work on automating RL model building and reducing the time to evaluate states, RL-guided DIDP has the potential to serve as a practical and powerful tool for combinatorial optimization.

References

1. Agostinelli, F., McAleer, S., Shmakov, A., Baldi, P.: Solving the rubik's cube with deep reinforcement learning and search. Nature Machine Intelligence **1**(8), 356–363 (2019). https://doi.org/10.1038/s42256-019-0070-z
2. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine **34**(6), 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240
3. Atamtürk, A., Narayanan, V.: Polymatroids and mean-risk minimization in discrete optimization. Operations Research Letters **36**(5), 618–622 (2008). https://doi.org/10.1016/j.orl.2008.04.006
4. Bellman, R.: Dynamic programming. Princetion University Press, Princeton, New Jersey (1957)
5. Bello*, I., Pham*, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. In: The Fifth International Conference on Learning Representations (2017), https://openreview.net/forum?id=rJY3vK9eg
6. Boisvert, L., Verhaeghe, H., Cappart, Q.: Towards a generic representation of combinatorial problems for learning-based approaches. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 99–108. Springer (2024). https://doi.org/10.1007/978-3-031-60597-0_7
7. Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P.: Combinatorial optimization and reasoning with graph neural networks. Journal of Machine Learning Research **24**(130), 1–61 (2023)
8. Cappart, Q., Moisan, T., Rousseau, L.M., Prémont-Schwarz, I., Cire, A.A.: Combining reinforcement learning and constraint programming for combinatorial optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 3677–3687. AAAI Press (2021). https://doi.org/10.1609/aaai.v35i5.16484
9. Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Advances in Neural Information Processing Systems. p. 6351–6361. Curran Associates Inc. (2017)
10. Flood, M.M.: The traveling-salesman problem. Operations research **4**(1), 61–75 (1956). https://doi.org/10.1287/opre.4.1.61
11. Fu, Z.H., Qiu, K.B., Zha, H.: Generalize a small pre-trained model to arbitrarily large TSP instances. In: Proceedings of the AAAI conference on artificial intelligence. vol. 35, pp. 7474–7482 (2021). https://doi.org/10.1609/aaai.v35i8.16916
12. Gehring, C., Asai, M., Chitnis, R., Silver, T., Kaelbling, L., Sohrabi, S., Katz, M.: Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 32, pp. 588–596 (2022). https://doi.org/10.1609/icaps.v32i1.19846
13. Huang, S., Ontañón, S.: A closer look at invalid action masking in policy gradient algorithms. arXiv preprint arXiv:2006.14171 (2020)
14. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996). https://doi.org/10.1613/jair.301
15. Kool, W., van Hoof, H., Gromicho, J., Welling, M.: Deep policy dynamic programming for vehicle routing problems. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 190–213. Springer (2022). https://doi.org/10.1007/978-3-031-08011-1_14

16. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=ByxBFsRqYm
17. Kuroiwa, R., Beck, J.C.: Domain-independent dynamic programming: Generic state space search for combinatorial optimization. In: Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS). vol. 33, pp. 236–244. AAAI Press (2023). https://doi.org/10.1609/icaps.v33i1.27200
18. Kuroiwa, R., Beck, J.C.: Solving domain-independent dynamic programming problems with anytime heuristic search. In: Proceedings of the 33rd International Conference on Automated Planning and Scheduling (ICAPS). AAAI Press (2023). https://doi.org/10.1609/icaps.v33i1.27201
19. Kuroiwa, R., Beck, J.C.: Domain-independent dynamic programming. arXiv preprint arXiv:2401.13883 (2024)
20. Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W.: Set transformer: A framework for attention-based permutation-invariant neural networks. In: Proceedings of the 36th International Conference on Machine Learning. vol. 97, pp. 3744–3753. PMLR (2019), https://proceedings.mlr.press/v97/lee19d.html
21. Li, S., Wang, R., Tang, M., Zhang, C.: Hierarchical reinforcement learning with advantage-based auxiliary rewards. In: Advances in Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA (2019)
22. Martello, S., Toth, P.: Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., USA (1990)
23. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on Machine Learning. vol. 48, pp. 1928–1937. PMLR (2016), https://proceedings.mlr.press/v48/mniha16.html
24. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015). https://doi.org/10.1038/nature14236
25. Nazari, M., Oroojlooy, A., Takáč, M., Snyder, L.V.: Reinforcement learning for solving the vehicle routing problem. In: Advances in Neural Information Processing Systems. vol. 31, p. 9861–9871. Curran Associates, Inc. (2018)
26. Orseau, L., Lelis, L., Lattimore, T., Weber, T.: Single-agent policy tree search with guarantees. In: Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
27. Orseau, L., Lelis, L.H.: Policy-guided heuristic search with guarantees. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 12382–12390. AAAI Press (2021). https://doi.org/10.1609/aaai.v35i14.17469
28. Puterman, M.L.: Markov decision processes. In: Handbooks in Operations Research and Management Science, vol. 2, pp. 331–434. Elsevier (1990)
29. Savelsbergh, M.W.: Local search in routing problems with time windows. Annals of Operations research **4**, 285–305 (1985). https://doi.org/10.1007/BF02022044
30. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: Proceedings of the 32nd International Conference on Machine Learning. vol. 37, pp. 1889–1897. PMLR (2015), https://proceedings.mlr.press/v37/schulman15.html
31. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

32. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–503 (2016). https://doi.org/10.1038/nature16961
33. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815 (2017)
34. Sun, Z., Yang, Y.: Difusco: Graph-based diffusion solvers for combinatorial optimization. In: Advances in Neural Information Processing Systems. vol. 36, pp. 3706–3731. Curran Associates Inc. (2023)
35. Sutton, R.S., Barto, A.G.: The reinforcement learning problem. In: Reinforcement Learning: An Introduction, pp. 51–85. MIT press (1998)
36. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations (2018), https://openreview.net/forum?id=rJXMpikCZ
37. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. In: Advances in Neural Information Processing Systems. vol. 30, pp. 3394–3404. Curran Associates Inc. (2017)
38. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Xu, C.: Learning to dispatch for job shop scheduling via deep reinforcement learning. In: Advances in Neural Information Processing Systems. vol. 33, pp. 1621–1632. Curran Associates Inc. (2020)