Learning to Plan with Tree Search via Deep RL

Dylan Cope[1,2], Justin Svegliato[2], Stuart Russell[2]
[1]King's College London
[2]University of California Berkeley
dylan.cope@kcl.ac.uk, jsvegliato@berkeley.edu, russell@berkeley.edu

Abstract

Tree search is an important component of many decision-making algorithms but often relies on an evaluation function that estimates the desirability of each node. In this paper, we propose to learn which nodes to expand based on a variety of object-level features. We introduce a reward function for this problem based on value of computation estimates with respect to improving the policy for the underlying problem. We apply deep reinforcement learning to this problem in an approach we call *Reinforcement Learning for Tree Search* (RLTS) and demonstrate that it can yield better performance than baselines in a procedurally generated environment.

1 Introduction

Planning algorithms that use tree search have been developed for many real-time decision-making problems, such as autonomous navigation (Koenig and Likhachev 2005), multi-robot coordination (Vedder and Biswas 2021), and trajectory optimization (Karaman et al. 2011). Simply put, these algorithms build a search tree from a start state to a goal state by expanding child nodes given an evaluation function. For instance, when selecting each child node to expand, heuristic search uses an estimate of the distance to the goal state (Hansen and Zhou 2007) while Monte Carlo tree search uses a measure that balances exploration and exploitation (Browne, Powley et al. 2012). However, while these algorithms rely on evaluation functions that reflect the desirability of each child node, there have been efforts to develop metareasoning approaches that explicitly reason about how to build a search tree.

A particularly promising approach to metareasoning was introduced by Russell and Wefald (1991). In order to optimize a notion of *time-dependent utility* that considers both the *value* of a solution and the *cost* of computation, they introduced a *meta-level decision problem* with computational states and actions that guides a planning algorithm over an *object-level decision problem*. Hence, a solution to the meta-level decision problem is a *meta-level policy*—a mapping from computational states to actions—that attempts to optimize time-dependent utility. Naturally, when this approach is instantiated in the context of tree search, the meta-level decision problem includes states for all possible search trees and actions for all possible node expansions.

While this approach offers a useful framework for metareasoning, it relies on exactly computing the time-dependent utility to find an optimal meta-level policy. This involves computing an expectation over all possible trajectories of computational states and actions, which is infeasible for planning algorithms that use tree search given the complexity of its space of search trees and node expansions. Hence, there have been several approaches that compute a meta-level policy by approximating time-dependent utility in some form (Hay et al. 2012; Callaway et al. 2018). However, while these approaches are effective in simple domains, they are challenging to scale up to large domains with continuous state and action spaces.

In this paper, we propose a novel approach that uses deep RL to optimize node expansion selection in tree search to maximize a reward function based on the value of computation (Matheson 1968; Russell and Wefald 1991). To do this, we introduce a meta-level decision problem that is constructed using a function $\hat{Q}^*(s, a)$ that estimates optimal object-level Q-values for states s and actions a. In our experiments, we show that our approach yields significant object-level performance gains over baseline techniques, especially with low quality initial Q-value estimates, on the BigFish Procgen benchmark (Cobbe et al. 2020).

The rest of this paper proceeds as follows. Sections 2, 3, and 4 define the meta-level/object-level decision problems, how to calculate policies from search trees, and how to estimate the value of computation to generate meta-level rewards. Sections 5 and 6 introduce our *Reinforcement Learning for Tree Search* (RLTS) approach and evaluate it on the BigFish Procgen benchmark. Section 7 concludes the paper.

2 Related Work

There has been a large body of work on metareasoning designed specifically for blackbox algorithms. First, there are methods that determine the optimal stopping point of the algorithm by tracking the performance of the algorithm and estimating its stopping point (Hansen and Zilberstein 2001; Lin et al. 2015; Svegliato, Wray, and Zilberstein 2018; Svegliato, Sharma, and Zilberstein 2020; Svegliato 2022). Next, there are methods that calculate the optimal parameter tuning of the algorithm based on properties of the specific problem at hand (Urmson and Simmons 2003; Hansen and Zhou 2007; Sun, Druzdzel, and Yuan 2007; Akgun and Stilman 2011; Kiesel, Burns, and Ruml 2012; Bhatia, Svegliato, and Zilberstein 2021). Finally, there are methods that use RL to learn to tune the parameters of the algorithm online (Biedenkapp et al. 2020; Bhatia et al. 2022).

Extending this work to multiple blackbox algorithms,

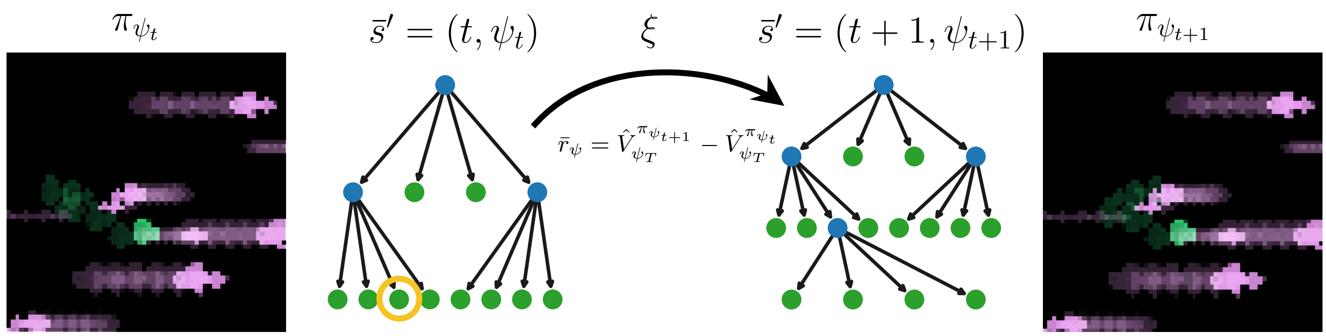

Figure 1: An illustration of applying an expansion 'computational action' ξ to a meta-level environment. On the left, at time t, we have the search tree ψ_t. Nodes that can be expanded are colored green, while those that cannot are blue. Each arrow represents taking a given action from the state specified in the parent node. The action ξ expands the node highlighted with the yellow circle, thereby the corresponding node in tree ψ_{t+1} has 4 children; one for each object-level action. We can also see representations of the greedy search tree policies π_{ψ_t} and $\pi_{\psi_{t+1}}$ at each time step, visualised as low-opacity overlay renders of the BigFish Procgen object-level environment. At time t, the agent's (the green fish's) plan is to traverse continually left and up. After the expansion, the plan is to go up and left until it reaches the small fish, at which point it can change to moving right and eat the other two fish. Finally, below the arrow indicating the action, we have an expression for the value-of-computation reward $\bar{r}_\Psi$ given for this expansion.

there are methods for optimal selection from an algorithm portfolio, recognizing that different algorithms tend to dominate each other on different problems. For example, there are methods that select the best algorithm for recovering from different exceptions or safety issues (Svegliato et al. 2019, 2022), the best algorithm for solving hard computational problems (Gomes and Selman 2001; Xu et al. 2008), and the best abstraction for solving Markov decision processees (Nashed et al. 2021, 2022). However, while these methods are designed for blackbox algorithms, our approach selects each individual computation to perform at runtime.

Metareasoning has also been applied to online planning (Gu et al. 2022) and situated temporal planning (Shperberg et al. 2020, 2021). Moreover, metareasoning has been used to learn search policies in a classical planning setting (Gomoluch et al. 2020). This work evaluates policies based on solving a search problem and optimizes policies with the cross-entropy method. In contrast, our approach solves the problem with reinforcement learning and evaluates search policies using value of computation estimates.

3 Meta-Level Decision Problem

We first define the *meta-level decision problem* that directs tree search over the *object-level decision problem*. Here, the goal is to find a meta-level policy that builds a search tree in a way that optimizes time-dependent utility, which involves selecting a *computational action* (node expansion) in each *computational state* (search tree). Finally, once this search tree is built, it is used to compute an object-level policy.

Definition 1. *The* ***object-level decision problem*** *is an MDP $M = \langle S, A, T, R, \gamma\rangle$, where S is the set of states, A is the set of actions, $T(s, a, s')$ is the transition function, $R(s, a, s')$ is the reward function, and γ is the discount factor.*

Given an object-level MDP $M = \langle S, A, T, R, \gamma\rangle$, we define a *search tree* $\psi \in \Psi$ as a recursive data structure comprised of *search tree nodes*. A node $n_s \in \mathrm{N}_\Psi$ is a tuple $(s, \mathcal{C})$, where $s \in S$ is an object-level state and $\mathcal{C}$ is a set of child nodes with the actions performed and the rewards observed when reaching the child: $(a, r, c) \in \mathcal{C} \subset A \times R \times \mathrm{N}_\Psi$.

A *node expansion* on a search tree node $n_s = (s, \mathcal{C}) \in \psi$ is an operation $\xi \in \Xi(\psi)$ that modifies n_s's children: $\mathcal{C} \leftarrow \mathcal{C} \cup \{(a, R(s, a, s'), (s', \emptyset)) : \forall a \in A, s' \sim T(s, a)\}$, which is denoted $\psi' = \xi \circ \psi$. Given a search tree ψ, we compute an object-level policy $\pi_\psi(a|s)$ with a process described later.

To perform tree search over the object-level decision problem, the meta-level decision problem either expands a node or terminates the search depending on whether the expected improvement to the object-level policy outweighs the cost of computation. Formally, we express this as the MDP below and illustrate a transition of the MDP in Figure 1.

Definition 2. *The* ***meta-level decision problem*** *that directs tree search over an object-level MDP M is an MDP $\bar{M} = \langle \Psi, \bar{S}, \bar{A}, \bar{T}, \bar{R}, \bar{\gamma}\rangle$:*

- *Ψ is a set of possible search trees over M.*
- *$\bar{S} = \mathbb{N} \times \Psi$ is a set of meta-level states: each computation state $\bar{s} = (t, \psi) \in \bar{S}$ indicates that tree search has the search tree $\psi \in \Psi$ after computing for $t \in \mathbb{N}$ time steps.*
- *$\bar{A} = \{\perp\} \cup \Xi$ is a set of possible meta-level actions: the* terminate action *$\perp$ terminates the search and each computation action $\xi \in \Xi(\psi)$ performs a node expansion on a tree $\psi \in \Psi$.*
- *$\bar{T} : \bar{S} \times \bar{A} \times \bar{S} \to [0, 1]$ is a transition function that represents the probability of reaching a state $\bar{s}' = (t', \psi') \in \bar{S}$ after performing an action $\xi \in \bar{A}$ in a state $\bar{s} = (t, \psi) \in \bar{S}$: the transition function $\bar{s}' \sim \bar{T}(\bar{s}' \mid \bar{s}, \xi)$ is*

$$\bar{s}' = \begin{cases} (0, \psi_{s'}) & \text{if } \xi = \perp \\ (t+1, \xi \circ \psi) & \text{otherwise} \end{cases}$$

  where $s' \sim T(s' \mid s,\ \pi_\psi(a|s))$, s is the object-level state at the root of ψ, and $\psi_{s'}$ is a root tree *with a single node.*
- *$\bar{R} : \bar{S} \times \bar{A} \times \bar{S} \to \mathbb{R}$ is the meta-level reward function $\bar{R}((t, \psi), \bar{a}, (t', \psi')) = \bar{R}_\Psi(\psi, \bar{a}, \psi') - C(t, \bar{a}, t')$, where $\bar{R}_\Psi$ is the reward for node expansions and C is the cost of node expansions.*
- *$\bar{\gamma}$ is the meta-level discount factor.*

4 Calculating Object-Level Policies

We now turn to how an object-level policy $\pi_\psi(a|s)$ is calculated from a search tree ψ.[1] The key idea is to use the information contained in the search tree ψ to calculate improved estimates of the optimal Q-values $\hat{Q}^*_\psi(s,a)$ for the object-level state-action pairs (s,a). Formally, starting with an *unconditioned* Q-value estimate $\hat{Q}^*(s,a)$, we define a *tree-conditioned* Q-value estimate $\hat{Q}^*_\psi(s,a)$ as follows:

$$\hat{Q}^*_\psi(s,a) = \begin{cases} r + \gamma \max_{a' \in A} \hat{Q}^*_\psi(s',a') & \exists\, (a,r,n_{s'}) \in \mathcal{C}^s_\psi \\ \hat{Q}^*(s,a) & \text{otherwise} \end{cases}$$

Note that $\mathcal{C}^s_\psi$ is the set of children from the node n_s in the tree ψ encoding object-level state s (if this node exists). Intuitively, this tree-conditioned Q-value estimate recursively applies Bellman back-ups using the trajectories observed in the tree until it reaches a leaf node. For each leaf node, it defaults to the unconditioned Q-value estimate.[2]

Naturally, a *greedy object-level policy* takes the optimal action with respect to the tree-conditioned Q-value estimate $\hat{Q}^*_\psi(s,a)$ (with ties broken randomly):

$$\pi_\psi(s) \in \arg\max_{a \in A} \hat{Q}^*_\psi(s,a)$$

We now show that the tree-conditioned Q-value estimate is constructed such that the optimal object-level policy can, in theory, be attained through tree search.

Theorem 1. *Given a deterministic object-level decision problem, there exists a meta-level policy $\bar{\pi}$ such that for any choice of $\hat{Q}^*$ the resulting tree ψ converges $\hat{Q}^*_\psi(s_{root},a)$ to $Q^*(s_{root},a)$ for all actions a.*

Proof (Sketch) 1. *Adapting the standard Bellman-optimality equation for Q^* to a deterministic environment simplifies to the expression in $\hat{Q}^*_\psi$ for the case that there exists a child node. Therefore, for a tree of infinite size where all nodes have children, $\hat{Q}^*_\psi = Q^*$. Further, for a tree where all nodes of depth d or less have children, we have that*

$$|Q^* - \hat{Q}^*_\psi| \leq \gamma^{d+1} \Big| \max_{s^l \in L_s} \max_{a' \in A} \Big(Q^*(s^l,a') - \hat{Q}^*(s^l,a') \Big) \Big|$$

where $s^l \in L_s$ is the set of states on leaf nodes. So, to converge towards the optimal Q-function, the meta-level policy $\bar{\pi}$ can perform breadth-first such, with the estimation error decreasing by $O(\gamma^d)$. Note that constructing a tree of depth d is expensive and requires $O(|A|^{d+1})$ computational steps.

5 Calculating Meta-Level Rewards

Our approach to designing the meta-level reward function $\bar{R}_\Psi$ is to incentivize selecting computational actions that are expected to be 'worthwhile' in that they have a high *value of computation* (VoC). Russell and Wefald (1991) define VoC as the difference in expected utility for performing the best action a_ζ after computation ζ and the expected utility for performing the best action a_0 before computation ζ: $\mathcal{V}(\zeta) = \mathbb{E}[U(a_\zeta)] - \mathbb{E}[U(a_0)]$. In practice, resource-bounded agents are not capable of computing these expected utilities, so they are instead estimates that depend on the computational state z of the agent: $\hat{\mathcal{V}}^z(\zeta) = \hat{\mathbb{E}}[U^z(a_\zeta)] - \hat{\mathbb{E}}[U^z(a_0)]$.

To apply this to our meta-level decision problem, we first define expected utilities as object-level value estimates. These are estimates of a policy π's expected discounted return from a state s, evaluated using the tree ψ:

$$\hat{V}^\pi_\psi(s) = \sum_{a \in A} \pi(a|s) \hat{Q}^\pi_\psi(s,a)$$

$$\hat{Q}^\pi_\psi(s,a) = \begin{cases} r + \gamma \hat{V}^\pi_\psi(s') & \exists\, (a,r,s') \in \mathcal{C}^s_\psi \\ \hat{Q}^*(s,a) & \text{otherwise} \end{cases}$$

Next, we need to compare the value estimates of the object-level policies π_{ψ_0} and π_{ψ_T} before and after a sequence of node expansions $\xi_1, \ldots, \xi_T$. These value estimates are computed with respect to the current state $s_{root} \in S$ of the object-level environment. Bringing this together results in the following expression for the estimated VoC over a series of node expansions:

$$\hat{\mathcal{V}}_{\psi_T}(\xi_1, \ldots, \xi_T) = \hat{V}^{\pi_{\psi_T}}_{\psi_T}(s_{root}) - \hat{V}^{\pi_{\psi_0}}_{\psi_T}(s_{root})$$

For brevity, we will denote the expected value estimates from the root states of the search tree policies at time t (evaluated using the tree at time T) as $\hat{V}^{\pi_{\psi_t}}_{\psi_T}(s_{root}) = \hat{V}^t_T$.

Finally, since the optimization objective of the meta-level environment is to maximise the value of computation over a series of node expansions $\hat{\mathcal{V}}(\xi_1, \ldots, \xi_T)$, the meta-level rewards of each computation is achieved by rearranging the VoC into a sum over each transition:

$$\hat{\mathcal{V}}_{\psi_T}(\xi_1, \ldots, \xi_T) = V^T_T - V^0_T = \sum_{t=1}^{T} \hat{V}^t_T - \hat{V}^{t-1}_T$$

$$\therefore\ \bar{R}_\Psi(\psi_{t-1}, \xi_t, \psi_t) = V^t_T - V^{t-1}_T$$

Note that computing each $\hat{V}^t_T$ in the expression for $\bar{R}_\Psi$ requires the search tree from the end of the meta-level episode, ψ_T. Therefore rewards can only be retroactively assigned to trajectories when the episode terminates.

Lemma 1. *The meta-level VoC return $\hat{\mathcal{V}}_{\psi_T}(\xi_1, \ldots, \xi_T)$ is always greater than or equal to zero.*

Proof (Sketch) 2. *This is equivalent to asserting that $V^T_T \geq V^0_T$. The policy π_{ψ_T} is greedy according to the Q-value estimates in ψ_T, meaning that by definition it has the maximum attainable value according to $\hat{V}_{\psi_T}$.*

This lemma tells us that a meta-level policy optimizing the VoC return will never be directly punished for performing computation. The metareasoning principle that the intrinsic utility is always positive is also satisfied, and thus the pressure to balance computation with resource limits can be controlled via the cost of computation function.

[1] In this paper and the experiments, we assume that the object-level decision problem is deterministic in the interest of simplicity.

[2] If object-level states appear multiple times in a search tree, aggregating Q-value estimates from each branch and handling cycles can improve $\hat{Q}^*_\psi$. In this paper, we omit these details as they are not relevant to the object-level decision problem in our experiments.

Lemma 2. *If the policy remains the same after the search, the value of computation is zero.*

Proof (Sketch) 3. *If* $\pi_{\psi_T} = \pi_{\psi_0}$ *then* $\hat{\mathcal{V}} = V_T^0 - V_T^0 = 0$.

6 Learning to Plan with Tree Search

In this section, we use deep RL to solve the meta-level decision problem, an approach we call *Reinforcement Learning for Tree Search* (RLTS). This requires three components: (1) a tensor representation of a search tree to provide observations to the neural network meta-policy $\bar{\pi}$; (2) a suitable DNN architecture for the meta-level policy $\bar{\pi}$; and (3) an unconditioned Q-value estimate $\hat{Q}^*$.

(1) The tensor representation of a search tree is a *tree tokenization*, where each node is expressed as a vector (*token*) of *structural* and *environmental* features. The key structural features include a node ID represented as a positional embedding of the node's index, the parent's node ID, and whether the node can be expanded. The key environmental features are the object-level reward observed after transitioning to the node, an encoding of the action performed to transition from the parent node to the node, the sum of object-level rewards generated along the trajectory leading to the node, the estimated discounted return attainable from the node's state ($\max_a \hat{Q}^*(s,a)$), and an encoding of the node's object-level state.

(2) The meta-level policy $\bar{\pi}$ is composed of a shallow *Transformer* (Vaswani et al. 2017), where the inputs are the search tree tokens and the outputs are a vector corresponding to meta-level actions. During training, we use *Proximal Policy Optimization* (PPO) (Schulman et al. 2017), which requires an actor network and a critic network (without weight sharing). For the actor network, we pass each of the transformer output vectors through a dense layer (with no activations) to result in an action logit for each of the corresponding meta-level actions. For the critic network, we sum together all of the transformer output vectors and use another dense layer to produce value estimates.

(3) The unconditioned Q-value estimate $\hat{Q}^*$ is learned by applying *RainbowDQN* (Hessel et al. 2018) to the object-level decision problem.

7 Procgen BigFish Experiments

In our experiments, we apply RLTS to the BigFish environment from the Procgen Benchmark (Cobbe et al. 2020). Procgen is considered here because procedurally generated environments are simple yet diverse, meaning that the agent cannot memorize a sequence of actions to succeed. Further, we used BigFish as it is one of the most reliable environments for training RainbowDQN. Our RainbowDQN model is a CNN with a penultimate hidden layer of 512 units. These vectors are extracted and used as state embeddings for the search tree tokens. We trained this Q-value estimate $\hat{Q}^*$ for 1.2M training steps and saved checkpoints of the model each time it improved in evaluation performance. The checkpoints are sorted so that we can pick the model at a given performance quantile, in order to have access to a variety of $\hat{Q}$ functions of differing quality. We trained four meta-level (RLTS) policies using $\hat{Q}^*$ checkpoints at the pretrained performance quantiles 0.25, 0.5, 0.75, and 0.9.

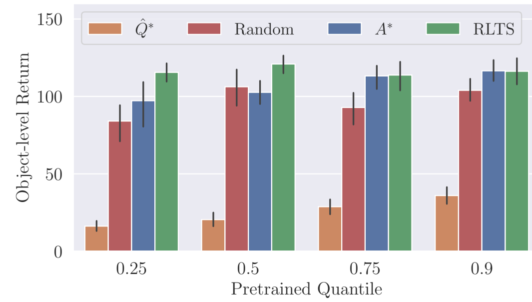

Figure 2: Mean object-level returns for different meta-level policies using pretrained models at varying performance quantiles.

In the meta-level decision problem over BigFish, we restrict the object-level action space to combinations of the arrow keys, resulting in 8 actions. All Procgen environments share the same set of 15 possible inputs, but 7 of these do not do anything in BigFish. This therefore reduces the branching factor of tree search from 15 to 8. The search trees are capped at a size of 64 nodes, meaning that if a policy attempts to expand beyond this limit the meta-level episode forcibly terminates. Upon termination with the final search tree ψ_T, an object-level action is sampled from π_{ψ_T} and taken in the object-level environment. The resulting state of the object-level environment is used as the root in a new search tree in and another meta-level episode commences.

To evaluate the learned policies for each pretrained performance quantile, we compare them to the greedy object-level baseline $\hat{Q}^*$ that does not perform any search and two meta-level baselines. The *random* meta-level baseline selects valid node expansions at random. The A^* meta-level baseline, a variant of the A^* heuristic search, selects valid node expansions that maximize $f(n) = g(n) + h(n)$, where g is the discounted sum of rewards observed from the root node to the node n and h is the discounted value estimate $\gamma^d \max_a \hat{Q}^*(s,a)$, where s is the object-level state associated with n. For each meta-level baseline, we measure the object-level return as it completes consecutive meta-level episodes and performs actions in BigFish. Across all evaluations, the random baseline gets a mean object-level return of 95.7, A^* gets 105.8, and RLTS gets $\mathbf{114.2}$. Figures 2 and 3 show the results of our experiments.

First, notice that all meta-level baselines significantly outperform the object-level baseline that greedily maximises over $\hat{Q}^*$. Even the random baseline, which generally performs the worst of the meta-level baselines, provides approximately 3-5x higher returns than without search. The A^* meta-level baseline steadily increases in performance inline with $\hat{Q}^*$. However, our RLTS approach consistently performs well across all pretrained performance quantiles. When $\hat{Q}^*$ is under-trained (quantiles 0.25 and 0.5), RLTS can compensate and outperforms the baselines. When $\hat{Q}^*$ is more refined (quantiles 0.75 and 0.9), RLTS and the A^* meta-level baseline have approximately equal performance.

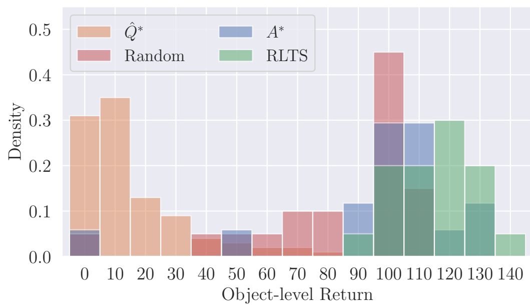

Figure 3: Distributions of object-level returns for each meta-level policy using the pretrained model at the 0.25 performance quantile.

Interestingly, the best performance is achieved when RLTS is applied to the 0.5 quantile. A hypothesis for this is that the more trained models are more confident in their estimates and less able to be 'persuaded' to change given the limited number of expansions. This is consistent with another observation that the mean meta-level return decreases as $\hat{Q}^*$ is trained. Recall that meta-level reward is given only if the policy changes (Lemma 2), and is awarded proportionally to how much better the new policy seems.

8 Conclusion

This paper offers a formalization of tree search as a meta-level decision problem and proposed a deep reinforcement learning approach. To provide a learning signal, we define the meta-level rewards using an estimate of the value of computation. We demonstrate that our approach outperforms hand-crafted methods and learn to compensate for under performing pretrained Q-networks. However, our work is limited to a straightforward environment and a short computational horizon. Future work will explore how to balance the trade-off between value and cost of computation.

References

Akgun, B.; and Stilman, M. 2011. Sampling heuristics for optimal motion planning in high dimensions. In *IROS*.

Bhatia, A.; Svegliato, J.; Nashed, S. B.; and Zilberstein, S. 2022. Tuning the hyperparameters of anytime planning: A metareasoning approach with deep reinforcement learning. In *ICAPS*.

Bhatia, A.; Svegliato, J.; and Zilberstein, S. 2021. On the benefits of randomly adjusting Anytime Weighted A*. In *12th SOCS*.

Biedenkapp, A.; Bozkurt, H. F.; Eimer, T.; Hutter, F.; and Lindauer, M. 2020. Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In *24th ECAI*.

Browne, C. B.; Powley, E.; et al. 2012. A survey of Monte Carlo tree search methods. *IEEE T-CIAIG*, 4(1).

Callaway, F.; Gul, S.; Krueger, P. M.; Griffiths, T. L.; and Lieder, F. 2018. Learning to select computations. In *34th UAI*.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2020. Leveraging procedural generation to benchmark RL. In *37th ICML*.

Gomes, C. P.; and Selman, B. 2001. Algorithm portfolios. *AIJ*, 126(1-2).

Gomoluch, P.; Alrajeh, D.; Russo, A.; et al. 2020. Learning neural search policies for classical planning. In *ICAPS*.

Gu, T.; Ruml, W.; Shperberg, S. S.; et al. 2022. When to commit to an action in online planning and search. *SOCS*.

Hansen, E. A.; and Zhou, R. 2007. Anytime heuristic search. *JAIR*, 28.

Hansen, E. A.; and Zilberstein, S. 2001. Monitoring and control of anytime algorithms. *AIJ*, 126(1-2).

Hay, N.; Russell, S.; Tolpin, D.; and Shimony, S. E. 2012. Selecting computations: Theory and applications. In *28th UAI*.

Hessel, M.; Modayil, J.; van Hasselt, H.; Schaul, T.; Ostrovski, G.; et al. 2018. Rainbow: Combining improvements in deep RL. In *AAAI*.

Karaman, S.; Walter, M. R.; Perez, A.; Frazzoli, E.; and Teller, S. 2011. Anytime motion planning using the RRT. In *ICRA*.

Kiesel, S.; Burns, E.; and Ruml, W. 2012. Abstraction-guided sampling for motion planning. In *5th SOCS*.

Koenig, S.; and Likhachev, M. 2005. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3).

Lin, C. H.; Kolobov, A.; Kamar, E.; and Horvitz, E. 2015. Metareasoning for planning under uncertainty. In *24th IJCAI*.

Matheson, J. E. 1968. The economic value of analysis and computation. *IEEE T-SSC*, 4(3).

Nashed, S. B.; Svegliato, J.; Bhatia, A.; Russell, S.; and Zilberstein, S. 2022. Selecting the partial state abstractions of MDPs: A metareasoning approach with deep reinforcement learning. In *IROS*.

Nashed, S. B.; Svegliato, J.; Brucato, M.; Basich, C.; Grupen, R.; and Zilberstein, S. 2021. Solving Markov decision processes with partial state abstractions. In *ICRA*.

Russell, S.; and Wefald, E. 1991. Principles of metareasoning. *AIJ*, 49.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms.

Shperberg, S.; Coles, A.; Karpas, E.; Shimony, E.; and Ruml, W. 2020. Trading plan cost for timeliness in situated temporal planning. In *29th IJCAI*.

Shperberg, S. S.; Coles, A.; Karpas, E.; Ruml, W.; and Shimony, S. E. 2021. Situated temporal planning using deadline-aware metareasoning. *ICAPS*.

Sun, X.; Druzdzel, M. J.; and Yuan, C. 2007. Dynamic Weighting A* search-based MAP algorithm for Bayesian networks. In *20th IJCAI*.

Svegliato, J. 2022. *Metareasoning for planning and execution in autonomous systems*. Ph.D. thesis, UMass Amherst.

Svegliato, J.; Basich, C.; Saisubramanian, S.; and Zilberstein, S. 2022. Metareasoning for safe decision making in autonomous systems. In *ICRA*.

Svegliato, J.; Sharma, P.; and Zilberstein, S. 2020. A model-free approach to meta-level control of anytime algorithms. In *ICRA*.

Svegliato, J.; Wray, K. H.; Witwicki, S. J.; Biswas, J.; and Zilberstein, S. 2019. Belief space metareasoning for exception recovery. In *IROS*.

Svegliato, J.; Wray, K. H.; and Zilberstein, S. 2018. Meta-level control of anytime algorithms with online performance prediction. In *27th IJCAI*.

Urmson, C.; and Simmons, R. 2003. Approaches for heuristically biasing RRT growth. In *IROS*.

Vaswani, A.; Shazeer, N.; Łukasz; Polosukhin, I.; et al. 2017. Attention is all you need. In *31st NeurIPS*.

Vedder, K.; and Biswas, J. 2021. X*: Anytime multi-agent path finding for sparse domains using window-based iterative repairs. *AIJ*, 291.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. *JAIR*, 32.