Exploring Long-Horizon Reasoning with Deep RL in Combinatorially Hard Tasks

Andrew C. Li [1 2] **Pashootan Vaezipoor** [1 2] **Rodrigo Toro Icarte** [3 4 2] **Sheila A. McIlraith** [1 2 5]

Abstract

Deep reinforcement learning has shown promise in discrete domains requiring complex reasoning, including games such as Chess, Go, and Hanabi. However, this type of reasoning is less often observed in long-horizon, continuous domains with high-dimensional observations, where instead RL research has predominantly focused on problems with simple high-level structure (e.g. opening a drawer or moving a robot as fast as possible). Inspired by combinatorially hard optimization problems, we propose a set of robotics tasks which admit many distinct solutions at the high-level, but require reasoning about states and rewards thousands of steps into the future for the best performance. Critically, while RL has traditionally suffered on complex, long-horizon tasks due to sparse rewards, our tasks are carefully designed to be solvable without specialized exploration. Nevertheless, our investigation finds that standard RL methods often neglect long-term effects due to discounting, while general-purpose hierarchical RL approaches struggle unless additional abstract domain knowledge can be exploited.

1. Introduction

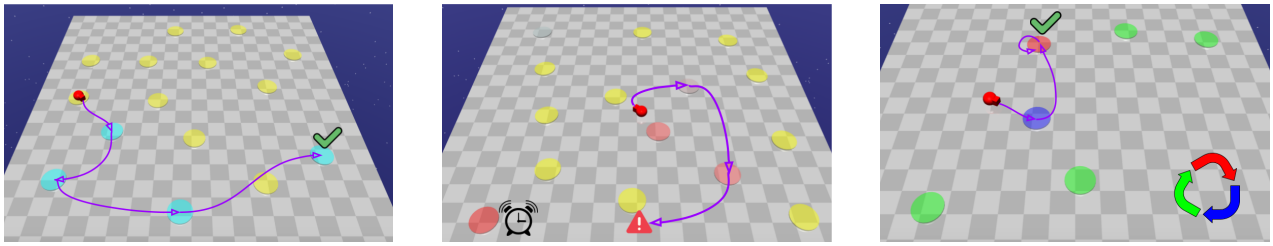
Reinforcement learning (RL) is a powerful framework for training sequential decision-making agents when the optimal behaviour is unknown or difficult to specify. Recently, RL has seen impressive strides across a number of domains requiring complex reasoning (e.g. game playing (Silver et al., 2018; Foerster et al., 2019; Anthony et al., 2017), combinatorial optimization (Mazyavkina et al., 2021; Kool et al., 2018)) as well as domains with long horizons and high-dimensional observations (Gupta et al., 2020; Duan et al., 2016; Tassa et al., 2018). Despite this, tasks which require a combination of high-level, temporally abstracted reasoning and low-level control over thousands of timesteps (encompassing many real-world problems) still remain a significant challenge for RL (Mirza et al., 2020). Often, this failure is due to reward sparsity — as tasks become increasingly long and difficult to solve, the reward signal also tends to diminish, making it impractical to learn from initially random behaviour.

Drawing inspiration from NP-hard combinatorial optimization problems, we contribute a suite of carefully designed tasks that can be decomposed to yield dense, shaped rewards with the following property: these tasks admit a vast number of feasible high-level solutions but solving them optimally requires the agent to reason over many distinct possible outcomes. We explore the prospect of solving such tasks with deep RL, and our contributions can be summarized as follows:

- Our investigation finds critical issues in applying deep RL to combinatorial RL tasks, even when sparse rewards are not problematic. These issues relate to the discount factor — undiscounted training is known to be unstable, while we show that discounted RL often leads to myopic behaviour inconsistent with maximizing long-term rewards.
- We propose a simple modification to PPO (Schulman et al., 2017) which drastically improves *undiscounted training* while achieving similar or better performance than vanilla PPO across our tasks.
- We further motivate hierarchical RL (HRL) for improved abstract reasoning, beyond typical sparse reward problems. We show that an HRL approach which exploits a hand-designed abstraction of the problem significantly outperforms standard (flat) deep RL. However, existing general-purpose HRL algorithms, while often claiming to produce meaningful temporal abstractions (Frans et al., 2017; Bacon et al., 2017), fail to provide these benefits.

[1]Department of Computer Science, University of Toronto [2]Vector Institute [3]Pontificia Universidad Católica de Chile [4]Centro Nacional de Inteligencia Artificial [5]Schwartz Reisman Institute for Technology and Society. Correspondence to: Andrew Li <andrewli@cs.toronto.edu>.

Decision Awareness in Reinforcement Learning Workshop at the 39th International Conference on Machine Learning (ICML), Baltimore, Maryland, USA, 2022. Copyright 2022 by the author(s).



(a) PointTSP (b) TimedTSP (c) ColourMatch

Figure 1. Combinatorial problems embedded in a MuJoCo robotics environment. **(a)** Visit all the zones as quickly as possible (blue = *to be visited*, yellow = *already visited*). A possible path is marked in purple. **(b)** Visit all zones (or as many as possible), but each zone is initialized with a timeout (darker red = *lower timeout*). Failing to visit any zone before it times out ends the episode (for the purple trajectory, the bottom-left zone timed out). **(c)** Make all zones the same colour. Visiting a zone cycles its colour in a fixed order (3 possible colours).

The rest of the paper is organized as follows. Section 2 provides an overview of related work. In Section 3, we describe our proposed combinatorial RL problems and how to encode them as non-sparse-reward MDPs. In Section 4, we investigate the prospect of solving these combinatorially hard, long-horizon tasks with standard deep RL, focusing on the important role of the discount factor. Section 5 explores whether the temporal abstractions afforded by hierarchy improve reasoning on these tasks. We refer to the background section in the Appendix for preliminary material on RL.

2. Related Work

Reinforcement learning has been successfully applied to a wide variety of domains involving complex reasoning. Such domains are often challenging due to a combinatorially large state space. In symbolic domains with known models — including games (e.g. Chess and Go) (Lai, 2015; Silver et al., 2018; Anthony et al., 2017; Foerster et al., 2019), theorem proving (Kaliszyk et al., 2018), symbolic regression (Landajuela et al., 2021), and combinatorial optimization problems (Mazyavkina et al., 2021) — methods typically rely on some form of search to improve reasoning. Recent RL research has also targeted discrete, combinatorally hard games with visual observations (Bagatella et al., 2021) such as Sokoban (Racanière et al., 2017). Unfortunately, combinatorially rich problems in complex, long-horizon environments generally pose significant issues for RL (Mirza et al., 2020) due to sparse rewards, with solutions often requiring additional assumptions, such as domain knowledge (Vaezipoor et al., 2021; León et al., 2020) or expert data (Guss et al., 2019). While we are interested more broadly in long-horizon problems that involve complex decision-making, our choice of tasks may evoke similarities to application areas such as robot motion planning (Mohanan & Salgoankar, 2018; Garrett et al., 2021). Indeed, motion planning problems also pose challenges in both long-term planning and low-level control. Traditionally, solutions often relied on domain knowledge (Latombe, 2012), but in recent years deep RL solutions have also been proposed (Aradi, 2020).

3. Combinatorial RL Tasks

In this section, we propose a suite of long-horizon, optimization-based robotics tasks[1] with underlying combinatorial structure (see Figure 1). Solving these tasks well requires the agent to learn both low-level motor control as well as long-term reasoning to optimize the performance criteria. We highlight two key properties of our domains before describing the specific tasks.

Combinatorial solution set: Each randomly generated instance from our proposed set of tasks admits a large number of distinct high-level feasible solutions. For example, in the PointTSP task (where the goal is to visit all "cities" on the grid as quickly as possible), there are 15! possible orderings in which the agent can visit the cities. Similar to NP-hard optimization tasks, we expect finding the optimal strategy to be challenging for most instances. Note that the optimal strategy depends on a number of factors, e.g., the placement of the cities, the speed and turning radius of the robotic agent, etc. This makes it difficult to handcraft strategies for these tasks. To ensure that agents cannot simply memorize well-performing solutions to a particular instance, we randomly generate a new map each episode (during training and evaluation), forcing the agent to learn a general strategy.

Reward density: In RL task design, it is often the case that rewards become increasingly sparse as we attempt to scale up the task complexity and horizon length (e.g. (Mirza et al., 2020; Duan et al., 2016; Yu et al., 2020)). While previous works have dealt with reward sparsity through specialized exploration (Eysenbach et al., 2018; Gregor et al., 2016), imitation learning (Hussein et al., 2017), or curriculum learn-

[1] Our environments are built off of OpenAI's Safety Gym (Ray et al., 2019) and our code is released at https://github.com/andrewli77/combinatorial-rl-tasks.

ing (Portelas et al., 2020), these methods may negatively bias the learned policy or require additional assumptions. Instead, dense rewards are a property of our tasks by design, allowing us to train policies via standard RL algorithms, tabula rasa.

3.1. PointTSP

Inspired by the NP-hard Travelling Salesman Problem (TSP), the objective of this task is to visit all 15 "cities", represented by zones, as quickly as possible (or if this is impossible within the time limit of 2000 steps, to visit as many cities). The robotic agent can accelerate and turn to navigate to these zones and observes its own position, velocity, the positions of zones (and their visitation status) and the time remaining in the episode. We assume for now that rewards are undiscounted (i.e. $\gamma = 1$, an assumption we discuss in Section 4). The goal of reaching all 15 zones as quickly as possible is encoded via a sparse reward of λt_{rem}, where $\lambda \in (0, \infty)$ is a hyperparameter (set to $\lambda = 0.01$ in our experiments), and t_{rem} is the time remaining when the task is solved. Observe that the reward is Markovian because the time remaining is observable to the agent. A dense reward of 1 is also provided whenever a new city is reached. Note that the dense reward is not misaligned with the original objective, since any successful trajectory garners a total return of $15 + \lambda t_{\text{rem}}$, while any unsuccessful trajectory garners a less than 15 total return. Thus, the agent is incentivized, first and foremost, to succeed in the task, and secondarily to succeed as fast as possible.

3.2. TimedTSP

This task introduces *timeouts* to PointTSP. Each zone $i \in \{1, ..., 15\}$ is randomly and independently initialized with a timeout c_i from a Beta distribution, observable to the agent. If any zone remains unvisited when its timeout expires, the episode ends and is considered unsuccessful. Thus, the agent may need to visit zones in an inefficient order (from the perspective of PointTSP) to prevent a single zone from expiring. Note that in some instances, a zone expiring may be imminent (e.g. if two far-apart zones are both about to expire), in which case it may be optimal to quickly visit as many zones as possible rather than trying to address the expiring zones.

3.3. ColourMatch

This environment contains six zones, each with a randomly initialized (observable) colour: red, green, or blue. Visiting a zone changes its colour to the next colour in a cycle, i.e. green $\rightarrow$ red $\rightarrow$ blue $\rightarrow$ green, and prevents it from being changed again for a short duration. The objective is to make all zones the same colour as quickly as possible. As before, a sparse reward of λt_{rem} is provided on the timestep the objective is met, incentivizing the agent to succeed as quickly as possible. To help decompose the task, we define the Hamming distance $H(s)$ in a state s as the minimum number of swaps required to solve the task. We provide a dense reward to the agent each time it changes the colour of a zone: for the transition $s \rightarrow s'$ where a zone colour change occurs, the dense reward would be $H(s) - H(s') \in \{1, 0, -1, -2\}$. This dense reward does not bias the original objective, since any successful trajectory from the initial state s_0 achieves the same total dense reward of $H(s_0)$. A valid strategy is to only cause colour changes that lead to positive dense rewards, solving the task in the fewest number of *colour swaps*. However, this may not optimally solve the task *as quickly as possible* since it does not consider other relevant factors, e.g., the location of the agent and zones.

3.4. Order-invariant Neural Architecture

Many combinatorial optimization problems are invariant to the order of their inputs, such as the order of jobs in a scheduling problem, or the order of graph edges in a TSP problem. We would also like to exploit this property in our domains. Consider as an example the PointTSP domain: the agent receives observations regarding the position and visitation status of zones, and the order in which the observations are conveyed is unimportant. A naive encoding that concatenates all state observations and feeds them into (say) a MLP loses this property of order invariance. This leads to sample inefficient learning as an agent does not recognize that a reward accrued by visiting a particular zone could have been gained by visiting *any* zone.

Instead, we encode the order-invariance of zones by borrowing ideas from Deep Sets networks (Zaheer et al., 2017; Lee et al., 2019). We treat the inputs related to the K zones as an unordered set $\{\mathbf{z}_1, ..., \mathbf{z}_K\}$, where each $\mathbf{z}_i$ is a fixed-sized vector consisting of the position and colour/visitation of that zone. All other aspects of the observation (agent's position and velocity, time remaining, etc) are denoted by a fixed-sized vector $\mathbf{x}$. We use an MLP layer f to jointly encode each zone $\mathbf{z}_i$ with the remaining observation $\mathbf{x}$ which is then aggregated over all zones via averaging, i.e. $\mathbf{w}^{(1)} = \frac{1}{n} \sum_{i=1}^{K} f(\text{Concat}(\mathbf{x}, \mathbf{z}_i))$. We then pass the aggregated vector $\mathbf{w}^{(1)}$ and $\mathbf{x}$ through an additional MLP layer g, allowing g to directly attend to relevant information in $\mathbf{x}$ (e.g. the time remaining) which may be difficult to otherwise capture in $\mathbf{w}^{(1)}$, i.e. $\mathbf{w}^{(2)} = g(\text{Concat}(\mathbf{w}^{(1)}, \mathbf{x}))$. $\mathbf{w}^{(2)}$ is then passed directly to the RL agent. All methods in our experiments used this order-invariant neural architecture to encode observations.

Finally, we remark that other observation modalities may naturally exploit order-invariance, such as images (i.e. the same image observation is generated regardless of the zone

ordering). However, learning to process image observations is often a strenuous task in itself and is detrimental to sample efficiency. This is not our main focus in the domains we explore and we instead opted for a lightweight representation of our observations.

4. The Role of Discounting

With the proposed long-horizon reasoning tasks encoded as MDPs, we now explore the prospect of solving these tasks directly with deep RL. We focus in particular on the discount factor $\gamma \in [0, 1]$, which plays a pivotal role by incentivizing immediate rewards over rewards in the future. Discounting is known to improve training stability (Amit et al., 2020; Schulman et al., 2015; Marbach & Tsitsiklis, 2003) and is nearly always applied in deep RL, even when the true goal is to maximize an *undiscounted* sum of rewards (Andrychowicz et al., 2020). Indeed, in our domains, which are finite-horizon (up to 2000 steps), the undiscounted objective is our goal. This assumption is made in Section 3 to ensure task rewards align with the goal of solving the task as quickly as possible.

While the benefits of discounting in deep RL are clear, it may unfortunately lead to a mismatch between the desired goal and the optimization objective. For example, when discounting at $\gamma = 0.99$, a common discount factor for MuJoCo tasks, a reward of 1 a thousand steps in the future is reduced to ≈ 0.00004, while our tasks last up to twice that many steps. To the best of our knowledge, domains demonstrating a significant objective mismatch due to discounting have thus far been limited to simple, tabular examples with known optimal policies (Naik et al., 2019; Mahadevan, 1996). In more complex domains, the optimal undiscounted return is rarely known due to the instability associated with training an undiscounted objective. We hypothesize that this *discounted objective mismatch* is especially critical to address in domains with long-term dependencies since an agent often must forego immediate rewards to maximize the long-term ones. The following experiments investigate this hypothesis.

Is discounting necessary for training? We first verify that discounting indeed improves performance on our proposed tasks, corroborating previous empirical studies that undiscounted training is unstable (Andrychowicz et al., 2020; Zhang et al., 2022). We trained PPO with discount factors $\gamma \in \{0.99, 0.9975, 1\}$ on each of the three proposed reasoning tasks. Final performances (as measured by *undiscounted* return) are reported in Figure 2. Training with lower discount factors improved the final performance on all domains, particularly TimedTSP. On PointTSP and TimedTSP, lower discount factors also significantly improved sample efficiency (as evidenced by the learning curves in the Appendix).

4.1. Training Without Discounting

We next discuss why undiscounted RL is so difficult and motivate a simple change to PPO which significantly improves performance in the undiscounted setting.

Value estimation with long horizons. We next consider how discounting affects *value estimation* with function approximation. In policy gradient methods, estimated values play a critical role in bootstrapping n-step returns and reducing variance as a baseline. Formally, we consider estimating the value function $V^{\pi,\gamma}(s) = \mathbb{E}_{\tau\sim\pi|s_0=s}[G^\gamma(\tau)]$, where $G^\gamma(\tau) = \sum_{i=1}^{H} \gamma^{i-1} r_i$ is the return of a trajectory τ (with rewards $r_1, ..., r_H$) under discount factor γ.

Due to stochasticity in the policy π and/or the environment dynamics, the return $G^\gamma(\tau)$ of a random trajectory starting from state s can have high variance. We emphasize that this variance often depends on the remaining horizon of s — states s with many remaining steps in the episode tend to have higher uncertainty in their returns. This can potentially be problematic at training time for several related reasons: (1) accurately estimating $V^{\pi,\gamma}(s)$ requires more samples; (2) the learned value function is more prone to overfitting to a poor batch of data, leading to instability; (3) states with high variance returns (e.g., those at the beginning of a long episode) may dominate the value loss, worsening the quality of predicted values for other states.

We demonstrate the severity of this variance in our environments. Figure 3 shows the empirical variance of $G^\gamma(\tau)$ from rolling out a fixed policy (obtained partway through training) from the same initial state 20 times. When evaluating $G^\gamma(\tau)$, we vary the discount factor $\gamma \in \{0.99, 0.9975, 1\}$ as well as the horizon length of the episode $H \in \{1, 2, ..., 2000\}$. With $\gamma = 1$, the variance of the return $G^\gamma(\tau)$ drastically increased with the horizon length while discounting with $\gamma = 0.99$ maintained a relatively low variance throughout the episode. This is expected as discounting effectively shortens the horizon by reducing the contributions of distant future rewards.

Dealing with high variance in value estimation. Besides discounting, value bootstrapping methods such as GAE (Schulman et al., 2015) are commonly used to trade-off between bias and variance in long-horizon problems. Unfortunately, even with such techniques, PPO trained poorly in the undiscounted setting.

Based on our previous observations, we propose and evaluate a simple modification to PPO to ameliorate the issue of high variance in $G^\gamma(\tau)$ when training without discounting. We consider learning the mean and variance of $G^\gamma(\tau)$, rather than a point estimate of the mean (as is typically done by the value network in PPO) for any state. This approach, which we refer to as PPO$_{\mathcal{VD}}$ ($\mathcal{VD}$ for "value distribution") is described in Algorithm 1. Intuitively, for states s where

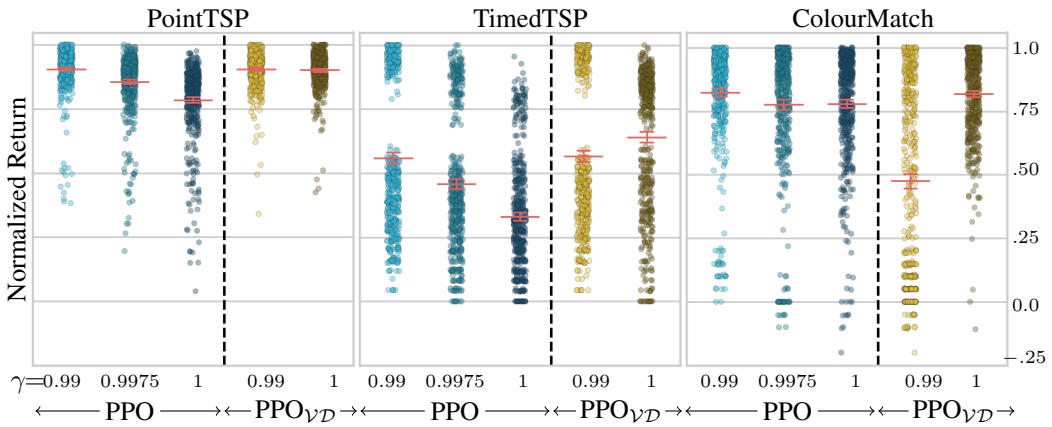


Figure 2. Final performances (normalized undiscounted returns) for PPO & PPO$_{\mathcal{VD}}$ trained with various discount factors. Returns are averaged over 5 policies $\times$ 100 random instances, then normalized by the best known performance for that instance. Error bars show 90% confidence intervals.

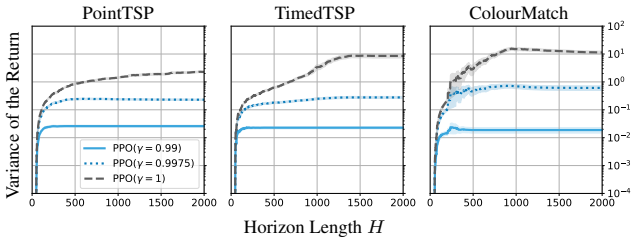


Figure 3. Variance of the return drastically increases with the horizon length in the undiscounted case. We show empirical variance in the return $G^{\gamma}(\tau)$ from 20 sampled trajectories of a fixed policy and initial state, for various horizon lengths and γ. Results are averaged over 20 instances $\times$ 5 policies, with error bars showing standard error. Note the log scale on the y-axis.

the return is predicted to have high variance, PPO$_{\mathcal{VD}}$ performs a more conservative update compared to a point estimate of the value, making PPO$_{\mathcal{VD}}$ less prone to overfitting within a batch of experience. While PPO$_{\mathcal{VD}}$ has similar motivations to distributional RL (Bellemare et al., 2017; Nam et al., 2021), we opt for a simpler setup, requiring minimal changes to the standard PPO algorithm. For example, we model the value distribution as Gaussian, which only requires learning a scalar mean and standard deviation, and the n-step returns (which use predicted values for bootstrapping) are treated as scalar targets rather than distributions. We are unaware of any prior work which learns value distributions to specifically address the instability of high discount factors in long-horizon environments.

The performance of PPO$_{\mathcal{VD}}$ is reported in Figure 2. A version of PPO$_{\mathcal{VD}}$ with $\gamma = 0.99$ is also included for comparison, though the algorithm is motivated for the undiscounted case. PPO$_{\mathcal{VD}}^{(\gamma=1)}$ converged to similar or better performance than PPO with discounting across all domains. Notably, on TimedTSP, PPO$_{\mathcal{VD}}^{(\gamma=1)}$ significantly outperformed PPO$^{(\gamma=0.99)}$ which we believe is due to optimizing the true objective – we provide further evidence of this below. Furthermore, PPO$_{\mathcal{VD}}^{(\gamma=1)}$ significantly outperformed PPO$^{(\gamma=1)}$, which we attribute to the improved stability of our approach in the undiscounted case. As expected, however, instability does not appear as problematic in the discounted setting: modelling a value distribution (i.e. PPO$_{\mathcal{VD}}^{(\gamma=0.99)}$) provided no additional benefit on top of PPO$^{(\gamma=0.99)}$ and was sometimes detrimental to performance. Lastly, discounting appears advantageous for sample efficiency: PPO$^{(\gamma=0.99)}$ was generally more sample efficient than PPO$_{\mathcal{VD}}^{(\gamma=1)}$.

Algorithm 1: PPO$_{\mathcal{VD}}$ (1 iteration). Changes with respect to standard PPO are marked in red.

Inputs: Number of steps T, discount factor γ, GAE parameter λ, number of optimization epochs K, policy network parameters θ, value network parameters ψ ;

`/* Data collection + processing */`

Collect T on-policy samples $(s_0, a_0, r_1, s_1, ..., a_T, r_T, s_{T+1})$ using π_θ;

Compute estimated values $v_t = \text{value_net}_\psi(s_t)$.mean;

Compute estimated advantages $\text{adv}_t = \sum_{l=0}^{T-t} (\gamma\lambda)^l (r_{t+l} + \gamma v_{t+l+1} - v_{t+l})$;

Compute value targets $\hat{v}_t = v_t + \text{adv}_t$;

`/* Policy optimization */`

for K *epochs* **do**

  Update θ using PPO loss and advantages;

  Update ψ using Gaussian negative-log-likelihood loss and value targets;

end

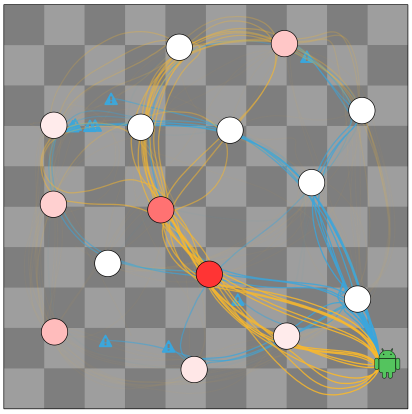


Figure 4. A visualization of 15 trajectories (5 policies $\times$ 3 rollouts) from PPO$^{(\gamma=0.99)}$ (*blue*) and PPO$^{(\gamma=1)}_{\mathcal{VD}}$ (*orange*) on a TimedTSP instance. Circles indicate zones, with darker red indicating lower timeout; the green robot indicates the starting position; X's indicate the end of a trajectory due to a timeout. Trajectories become more transparent over time to focus on each policy's initial behaviour. PPO$^{(\gamma=1)}_{\mathcal{VD}}$ immediately visits the two zones with lowest timeout, while PPO$^{(\gamma=0.99)}$ neglects this and quickly fails.

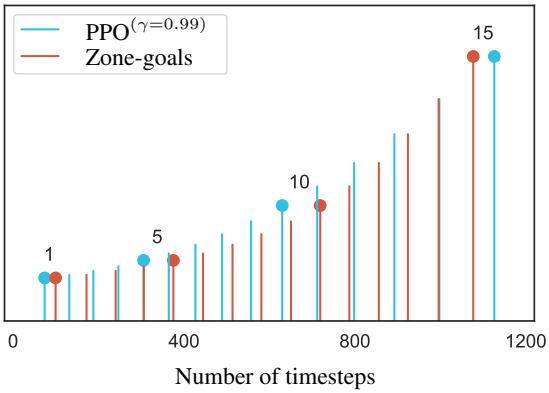


Figure 5. The cumulative time spent in PointTSP to visit i zones, for $i \in \{1, 2, ..., 15\}$ (averaged over 5 policies $\times$ 100 instances). The height of each line increases with i, and lines corresponding to $i \in \{1, 5, 10, 15\}$ are marked with a circle. PPO$^{(\gamma=0.99)}$ rapidly visits the first few zones, but spends more time visiting all 15 zones than Zone-goals.

How does objective mismatch due to discounting affect performance? We previously observed that optimizing the *undiscounted* objective with PPO$_{\mathcal{VD}}$ can improve performance over discounted RL. Here we take a closer look at how discounting can be detrimental to decision-making. We report in Table 1 the final performances of PPO$^{(\gamma=0.99)}$, PPO$^{(\gamma=1)}_{\mathcal{VD}}$, and Zone-goals$^{(\gamma=1)}$ (a strong baseline we introduce in the next section to optimize the undiscounted objective) using both discounted ($\gamma = 0.99$) and undiscounted ($\gamma = 1$) returns as the evaluation criteria. We find evidence of objective mismatch due to discounting on TimedTSP and, to a lesser extent, on PointTSP — Zone-goals simultaneously performed better than PPO$^{(\gamma=0.99)}$ when evaluated with $\gamma = 1$ but also worse when evaluated with $\gamma = 0.99$.

On TimedTSP, we observed critical differences in the decision-making of PPO$^{(\gamma=0.99)}$ vs PPO$^{(\gamma=1)}_{\mathcal{VD}}$. A visualization of each method's behaviour on a challenging instance is shown in Figure 4. In this instance, two zones are initially close to timing out and must be visited promptly to continue the episode. All trajectories from PPO$^{(\gamma=1)}_{\mathcal{VD}}$ immediately visited these zones and managed to achieve a high return while most trajectories from PPO$^{(\gamma=0.99)}$ neglected these zones and quickly failed. Notably, PPO$^{(\gamma=0.99)}$ always greedily visited the closest zone to the agent's starting position first, while none of the PPO$^{(\gamma=1)}_{\mathcal{VD}}$ trajectories did.

	PointTSP		TimedTSP		ColourMatch	
baseline γ_{eval}	0.99	1	0.99	1	0.99	1
PPO$^{(\gamma=0.99)}$	**1.13**	23.48	**1.12**	14.15	0.60	16.45
PPO$^{(\gamma=1)}_{\mathcal{VD}}$	0.66	23.36	0.67	16.19	0.32	16.33
Zone-goals$^{(\gamma=1)}$	0.81	**24.24**	0.91	**21.95**	**0.93**	**18.95**

Table 1. On PointTSP and TimedTSP, the best baseline depends on γ_{eval} showing that the $\gamma = 1$ and $\gamma = 0.99$ objectives are not aligned. The average return of 500 (5 policies $\times$ 100 instances) trajectories from each approach is reported.

While less pernicious towards performance, discounted PPO also showed signs of myopic decision-making on PointTSP. We observed that PPO$^{(\gamma=0.99)}$ often left outlying cities unvisited until the end in order to more rapidly visit cities at the start. This is evidenced in Figure 5, where we report the average cumulative time taken to visit i cities (for each i up to 15) for PPO$^{(\gamma=0.99)}$ and Zone-goals. Despite visiting the first few cities quicker than Zone-goals, PPO$^{(\gamma=0.99)}$ spends considerable time visiting the last few cities and ultimately is slower in solving the task.

On ColourMatch, we remark that Zone-goals outperformed PPO$^{(\gamma=0.99)}$ on both the discounted *and* undiscounted objectives, implying that discounting cannot be the only source of suboptimality. We believe this is since ColourMatch presents a more difficult reasoning task – in fact, even determining whether visiting a zone will result in a positive reward is non-trivial as it depends on the colours of the other zones.

5. The Role of Hierarchy

In the previous section, we observed significant issues with PPO in long-horizon reasoning tasks: training with γ close to 1 was often unstable while lowering the discount factor exhibited myopic behaviour. Here, we turn our attention to hierarchical reinforcement learning (HRL), which has widely been applied to long-horizon tasks in the past. HRL

is often motivated by the hierarchical structure present in real-world tasks — to make pizza, one must roll out the dough, prepare the sauce, add toppings, and bake the pizza. Unfortunately, as tasks increase in length and complexity, they quickly become unsolvable by methods which learn from random initial behaviour in the sparse reward setting. Many works in deep HRL aim to address this reward sparsity by improving exploration (e.g. (Eysenbach et al., 2018; Li et al., 2019; Florensa et al., 2017; Zhang et al., 2021)). Indeed, mitigating the impact of sparse rewards has been cited as the main benefit resulting from hierarchy in existing methods (Nachum et al., 2019; Jong et al., 2008).

Here, we attempt to motivate a different rationale for the use of hierarchy. In our long-horizon environments, single actions are unlikely to lead to meaningfully different states and therefore, exploiting the high-level structure in our tasks may require reasoning over concepts that are temporally abstracted. In this section, we investigate whether the temporal abstractions afforded by HRL indeed improve reasoning on combinatorial RL tasks compared to non-hierarchical RL.

5.1. Experimental Setup

A wide variety of HRL algorithms have previously been proposed in the literature and we attempt to cover a representative sample of them here. We restrict ourselves to the two-level hierarchical framework, as is most common in deep RL environments. In this framework, a high-level policy operates at a lower temporal frequency by selecting high-level actions which persist for many timesteps (either a fixed number, or based on state-dependent termination criteria). A low-level policy conditions on the current high-level action to select actions in the environment every timestep. We assume both high and low-level policies receive the full environment observation. The manner in which the high and low-level policies are trained also varies between algorithms. In each of our HRL implementations, we concurrently train both the high and low-level policies with PPO, using the same on-policy data to optimize each, similar to (Li et al., 2019). While off-policy HRL approaches exist (Nachum et al., 2018) and are potentially more sample efficient, they are generally less stable due to the worsened effects of non-stationarity from training both policies simultaneously. All methods were trained for 100 million frames on PointTSP and 150 million frames on TimedTSP and ColourMatch, and the hidden layer sizes were adjusted such that each method used roughly the same number of parameters. The complete hyperparameter settings are reported in the Appendix.

5.2. Methods and Results

We describe the HRL methods we considered and report their performances on our suite of long-horizon reasoning tasks in Figure 6.

Fixed-length skills (Florensa et al., 2017; Frans et al., 2017): A simple and common HRL approach uses the high-level policy to choose a discrete low-level skill to execute every k timesteps, where k is a constant. Both the high and low-level policies are optimized directly using the environment reward. We choose to optimize the low-level policy using *discounted* environment rewards as learning was unstable with undiscounted rewards. However, as the high-level policy essentially faces a shorter horizon length due to actions being temporally extended, we optimize the high-level policy using *undiscounted* environment rewards to avoid the issue of objective mismatch.

Result. Fixed-length skills achieved similar performance to flat PPO. Even when evaluated with random skills, the performance did not drop, suggesting that the different skills became indistinguishable from one another. In such cases, the HRL policy effectively collapses into a flat policy and is unable to benefit from temporally extended actions.

DIAYN (Eysenbach et al., 2018): To avoid learning a homogeneous set of skills, we consider applying an information-theoretic diversity objective to the fixed-length skills framework. Such approaches typically learn task-agnostic skills in a reward-free setting to improve downstream exploration. Instead, we explore whether promoting skill diversity in our domains can aid in learning more meaningful high-level actions. We linearly combine task rewards with the DIAYN objective when training the low-level policy: $r'_t = r_t + \alpha(\log q_\phi(z_t|s_{t+1}) - \log p_\psi(z_t))$, where $\alpha \in [0, \infty)$ is a hyperparameter, r_t the environment reward, z_t the current discrete skill, q_ϕ a neural network trained to predict the skill distribution given the next state, and p_ψ a neural network trained to predict the prior skill distribution. The high-level policy was trained to maximize undiscounted environment rewards (without the diversity objective).

Result. The inclusion of the diversity term performed worse than fixed-length skills in all tasks. Thus, simultaneously optimizing for diversity and reward appears detrimental to reward.

Options (Sutton et al., 1999; Bacon et al., 2017): An *option* ω is a temporally extended action which consists of a low-level policy $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0, 1]$, a set of states $\mathcal{I}_\omega \subseteq \mathcal{S}$ where ω can be initiated, and a termination condition $\beta_\omega : \mathcal{S} \to [0, 1]$. When an option terminates, a high-level policy chooses the next option from among those that can be initiated. To simplify learning, we assume all options can be initiated in any state (similar to (Bacon et al., 2017)). We learn the termination condition by augmenting the action-space of the low-level policy with a special `stop` action. As before, the low-level policy and the high-level policy over options are trained on the discounted and undiscounted (resp.) environment rewards.

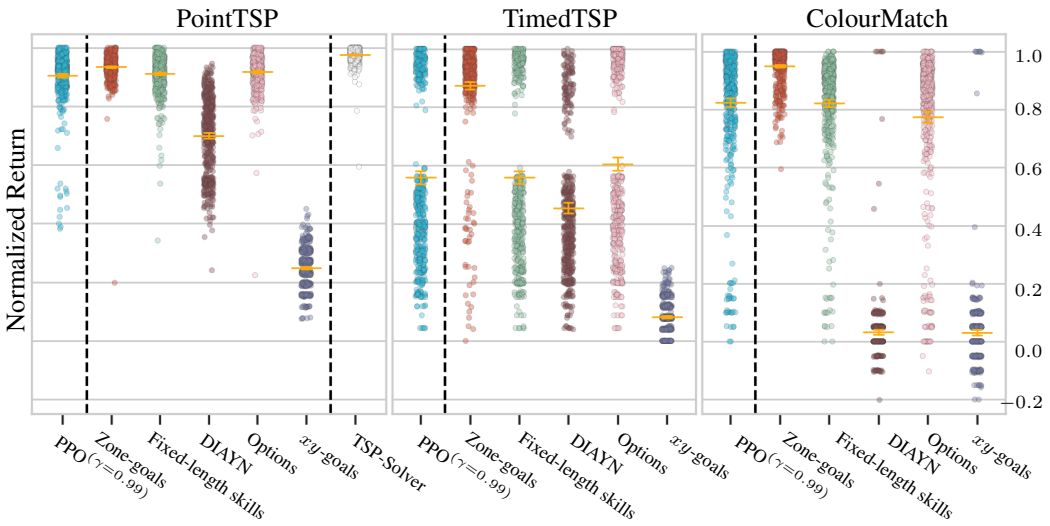


Figure 6. Final performances (normalized undiscounted returns) for various HRL methods. Returns are averaged over 5 policies $\times$ 100 random instances, then normalized by the best known performance for that instance. Error bars show 90% confidence intervals.

Result. Learning the termination condition provided similar performance to fixed-length skills. We observed that the learned skills usually terminated after fewer than 10 steps, making it difficult for the high-level policy to meaningfully reason over long horizons.

***xy*-goals:** Inspired by goal-based HRL (Gürtler et al., 2021; Nachum et al., 2018; Chane-Sane et al., 2021; Vezhnevets et al., 2017), we evaluate an approach where the high-level policy assigns xy-goals $g \in \mathbb{R}^2$ to the low-level agent. The low-level policy receives only a dense reward based on the change in Euclidean distance to the goal every timestep while the high-level policy directly receives rewards from the environment.

Result. This approach performed poorly on all domains. Since the low-level policy does not directly receive task rewards, it instead must rely on the goals set by the high-level policy to solve the task. xy-goals do not appear well-suited for these tasks, as we observed the agent often missed important zones despite reaching the target locations effectively. Neither increasing the frequency of goal selection nor using full states as goals (similar in spirit to (Nachum et al., 2018)) improved performance.

Zone-goals (Ours): We design a domain-specific goal-space based on the zones in the environment. With the current observation, the high-level policy produces a score for each zone and selects the next zone the agent should visit using a softmax over the scores. Certain zones in PointTSP and TimedTSP are invalid choices if previously visited, and are masked out, with the policy gradients adjusted accordingly (Tang et al., 2020). The low-level policy receives the xy-coordinates of the goal zone in addition to the current observation, and is rewarded in the same manner as the xy-goals approach.

Result. This method significantly outperformed the previous approaches (including standard PPO), particularly on TimedTSP and ColourMatch. This demonstrates the efficacy of hierarchy towards long-term reasoning, given the proper abstraction of the problem. Despite optimizing the undiscounted reward at the high-level, the instability observed in standard PPO in Section 4 did not present here.

TSP-Solver (Ours): On the PointTSP domain, we consider generating a open-loop, high-level plan at the start of every episode over the order in which zones should be visited. The plan is generated using a metric TSP solver with the xy-coordinates of zones as city locations. With this fixed ordering of zones (which replaces the role of the high-level policy), the low-level policy is rewarded based on xy-distance towards the next unvisited zone. However, we allow the low-level policy to plan ahead by observing the entire ordering of zones — for the i-th zone in the ordering, the observation for that zone is augmented with a scalar value of 2^{-i+1} to uniquely identify its position in the ordering.

Result. Using a planner outperformed all other approaches on PointTSP, showing the advantage of abstract planning in complex, long-horizon domains. We note that the plan generated is not necessarily optimal, since the solver does not consider the agent's acceleration, turning speed, and other relevant factors to the problem.

6. Conclusion

In this work we studied long-horizon robotics tasks that are combinatorially hard but that afford a number of feasible solutions. To this end, we proposed a representative set of tasks that challenge the performance of state-of-the-art RL

methods. Our tasks involve both low-level motor control as well as complex, long-term reasoning about delayed rewards and distinct possibilities over future states. Crucially, these tasks can be solved (albeit suboptimally) by standard RL methods without specialized exploration. Through our investigation, we uncovered several weaknesses with state-of-the-art RL algorithms in the context of long-term reasoning, highlighting promising directions for future work.

Undiscounted RL. Discounting is nearly always applied in long-horizon deep RL problems for stability, however this can cause a significant objective mismatch in some of our domains. We showed that this leads to "myopic" decisions which fail to consider delayed rewards. Thus far, undiscounted RL (or average-reward RL) has received little attention from the deep RL community. We believe that improving the robustness and sample efficiency for undiscounted RL will become increasing important as RL scales to longer and more complex tasks.

Hierarchical RL. While the benefits of hierarchy in planning and reasoning are often espoused (Erol et al., 1994; Sutton et al., 1999; Currie & Tate, 1991), most HRL research to date has focused on the challenge of exploration. Instead, we motivate the use of hierarchy as a way to improve long-term reasoning where standard RL methods fail. We demonstrated in our combinatorial RL domains that leveraging the proper hierarchical abstraction can substantially improve convergence performance over standard RL. However, this hierarchical abstraction was built using domain knowledge of the problem, and existing general-purpose HRL approaches were unable to provide similar benefits over PPO. In particular, hierarchies based on learned skills were prone to collapsing into a single skill, nullifying any benefits over flat policies. Thus, the automated discovery of hierarchies suitable for temporally extended reasoning and planning remains an important problem for future work.

In conclusion, we presented long-horizon RL problems that involve combinatorial reasoning to scale up task complexity while avoiding the issue of sparse rewards, which has traditionally plagued RL task design. We believe these types of problems can expedite future RL research due to the unique challenges they pose to existing RL methods.

7. Acknowledgements

We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, and Microsoft Research. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute for Artificial Intelligence (www.vectorinstitute.ai/partners). Finally, we thank the Schwartz Reisman Institute for Technology and Society for providing a rich multi-disciplinary research environment.

References

Amit, R., Meir, R., and Ciosek, K. Discount factor as a regularizer in reinforcement learning. In *International conference on machine learning*, pp. 269–278. PMLR, 2020.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*, 2020.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. *Advances in Neural Information Processing Systems*, 30, 2017.

Aradi, S. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Bagatella, M., Olšák, M., Rolínek, M., and Martius, G. Planning from pixels in environments with combinatorially hard search spaces. *Advances in Neural Information Processing Systems*, 34, 2021.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.

Chane-Sane, E., Schmid, C., and Laptev, I. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pp. 1430–1440. PMLR, 2021.

Currie, K. and Tate, A. O-plan: The open planning architecture. *Artificial Intelligence*, 52 (1):49–86, 1991. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(91)90024-E. URL https://www.sciencedirect.com/science/article/pii/000437029190024E.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pp. 1329–1338. PMLR, 2016.

Erol, K., Hendler, J. A., and Nau, D. S. Umcp: A sound and complete procedure for hierarchical task-network planning. In *Aips*, volume 94, pp. 249–254, 1994.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.

Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

Foerster, J., Song, F., Hughes, E., Burch, N., Dunning, I., Whiteson, S., Botvinick, M., and Bowling, M. Bayesian action decoder for deep multi-agent reinforcement learning. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1942–1951. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/foerster19a.html.

Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., and Lozano-Pérez, T. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, pp. 1025–1037. PMLR, 2020.

Gürtler, N., Büchler, D., and Martius, G. Hierarchical reinforcement learning with timed subgoals. *Advances in Neural Information Processing Systems*, 34, 2021.

Guss, W. H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., and Salakhutdinov, R. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

Jong, N. K., Hester, T., and Stone, P. The utility of temporal abstraction in reinforcement learning. In *AAMAS (1)*, pp. 299–306. Citeseer, 2008.

Kaliszyk, C., Urban, J., Michalewski, H., and Olšák, M. Reinforcement learning of theorem proving. *Advances in Neural Information Processing Systems*, 31, 2018.

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

Lai, M. Giraffe: Using deep reinforcement learning to play chess. *arXiv preprint arXiv:1509.01549*, 2015.

Landajuela, M., Petersen, B. K., Kim, S., Santiago, C. P., Glatt, R., Mundhenk, N., Pettit, J. F., and Faissol, D. Discovering symbolic policies with deep reinforcement learning. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5979–5989. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/landajuela21a.html.

Latombe, J.-C. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pp. 3744–3753. PMLR, 2019.

León, B. G., Shanahan, M., and Belardinelli, F. Systematic generalisation through task temporal logic and deep reinforcement learning. *arXiv preprint arXiv:2006.08767*, 2020.

Li, S., Wang, R., Tang, M., and Zhang, C. Hierarchical reinforcement learning with advantage-based auxiliary rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

Mahadevan, S. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22(1):159–195, 1996.

Marbach, P. and Tsitsiklis, J. N. Approximate gradient methods in policy-space optimization of markov reward processes. *Discrete Event Dynamic Systems*, 13(1):111–148, 2003.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134: 105400, 2021.

Mirza, M., Jaegle, A., Hunt, J. J., Guez, A., Tunyasuvunakool, S., Muldal, A., Weber, T., Karkus, P., Racanière, S., Buesing, L., et al. Physically embedded planning problems: New challenges for reinforcement learning. *arXiv preprint arXiv:2009.05524*, 2020.

Mohanan, M. and Salgoankar, A. A survey of robotic motion planning in dynamic environments. *Robotics and Autonomous Systems*, 100:171–185, 2018.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019.

Naik, A., Shariff, R., Yasui, N., Yao, H., and Sutton, R. S. Discounted reinforcement learning is not an optimization problem. *arXiv preprint arXiv:1910.02140*, 2019.

Nam, D. W., Kim, Y., and Park, C. Y. Gmac: A distributional perspective on actor-critic framework. In *International Conference on Machine Learning*, pp. 7927–7936. PMLR, 2021.

Portelas, R., Colas, C., Weng, L., Hofmann, K., and Oudeyer, P.-Y. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. Imagination-augmented agents for deep reinforcement learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/9e82757e9a1c12cb710ad680db11f6f1-Paper.pdf.

Ray, A., Achiam, J., and Amodei, D. Benchmarking Safe Exploration in Deep Reinforcement Learning. 2019.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/abs/10.1126/science.aar6404.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Tang, C.-Y., Liu, C.-H., Chen, W.-K., and You, S. D. Implementing action mask in proximal policy optimization (ppo) algorithm. *ICT Express*, 6(3):200–203, 2020.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Vaezipoor, P., Li, A. C., Icarte, R. A. T., and Mcilraith, S. A. LTL2Action: Generalizing LTL instructions for multi-task RL. In *International Conference on Machine Learning*, pp. 10497–10508. PMLR, 2021.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 3540–3549. PMLR, 2017.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100. PMLR, 2020.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhang, J., Yu, H., and Xu, W. Hierarchical reinforcement learning by discovering intrinsic options. In *9th International Conference on Learning Representations, ICLR*, 2021.

Zhang, S., Laroche, R., van Seijen, H., Whiteson, S., and des Combes, R. T. A deeper look at discounting mismatch in actor-critic algorithms. In *21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, pp. 1491–1499, 2022.

Appendices

A. Background

In this paper, we consider the discounted formulation of a Markov Decision Process (MDP) $\mathcal{M} = \langle S, T, A, p, r, \gamma, \mu \rangle$, where S is a set of *states*, $T \subseteq S$ a set of *terminal states*, A a set of *actions*, $p(s'|s,a)$ the *transition probability distribution*, $r(s,a,s')$ the *reward function*, γ the *discount factor*, and μ the *initial state distribution*. For each *episode*, the initial state s_0 is randomly sampled from μ. The agent then chooses actions a_t at each timestep $t \geq 0$ according to its policy $\pi(a_t|s_t)$ and receives the next state s_{t+1} according to transition probabilities (also known as the *dynamics*) $p(s_{t+1}|s_t, a_t)$ and a reward r_{t+1} according to $r(s_t, a_t, s_{t+1})$. This proceeds until a terminal state in T is reached. The *discounted return* in an episode of length H is defined as $\sum_{i=1}^{H} \gamma^{i-1} r_i$ and the goal of the agent is to learn a policy $\pi(a|s)$ that maximizes the expected discounted return.

We focus in particular on *on-policy policy gradient methods*, a popular class of RL algorithms for environments with continuous actions and states. Here, the policy $\pi_\theta(a|s)$ is stochastic and parametrized by θ (e.g. θ could be the weights of a neural network). On-policy methods sample data from the environment according to the current policy $\pi_\theta(a|s)$ and attempt to directly estimate the gradient of the expected return $\nabla_\theta \mathbb{E}[\sum_{i=1}^{H} \gamma^{i-1} r_i]$, where the expectation is over the stochasticity of the policy and the environment transitions. The estimated gradients are then used to update the policy parameters θ (hopefully resulting in a higher expected return) using gradient ascent.

B. Packages and Licensing

The code for our proposed set of environments, along with scripts for training and evaluation will be publicly released upon publication. Our environments are built off of the OpenAI Safety Gym[2] while the RL training code is built off of the torch-ac repository[3], both of which are available under an MIT license. Note that the OpenAI Safety Gym uses MuJoCo [4] which is freely available under an Apache 2.0 license.

C. Additional Experimental Details

All experiments were conducted on a compute cluster using a single GPU and 16 CPU cores per run. PPO training lasted roughly 18 hours for PointTSP (for 100 million frames), 55 hours for PointTSP (for 150 million frames), and 27 hours for ColourMatch (for 150 million frames). Other methods required a similar amount of training time, except for Options, which required about 50% more time. Note that with fixed-length skills, calls to the high-level policy could be synchronized across parallel training environments, while this was not possible with variable-length Options.

We show the learning curves of PPO and HRL baselines in Figures 7 and 8 respectively.

C.1. Network Architectures

Here, we report the network architectures for all methods in our experiments. The sizes of hidden layers are reported in the next section and were chosen to ensure the number of parameters of each method were roughly equal. All activations are ReLU.

PPO$^{(\gamma=0.99)}$, PPO$^{(\gamma=0.9975)}$, PPO$^{(\gamma=1)}$, PPO$^{(\gamma=0.99)}_{\mathcal{VD}}$, PPO$^{(\gamma=1)}_{\mathcal{VD}}$: Observations were first encoded with the order-invariant neural architecture described in Section 3.4: each zone is encoded using an MLP with two hidden layers, and then the average of all zone encodings is aggregated through a single linear layer. This observation encoding is then passed to the actor and critic (both single hidden-layer MLPs). For PPO$_{\mathcal{VD}}$, the final critic layer has two output heads to model a mean and standard deviation (using softplus for the latter).

Fixed-length skills, DIAYN, Options: The high and low-level policies each use the same architecture as for PPO with minor changes: the hidden sizes are reduced to maintain the same number of parameters; the high-level policy outputs a discrete skill (5 possible values); and the low-level policy conditions on the current skill. For Options, the low-level policy additionally outputs a probability of terminating the current skill. For DIAYN, we additionally train a neural network to predict the current skill given next state. This network uses the order-invariant neural architecture: a two hidden-layer MLP for encoding zones and a one hidden-layer MLP for aggregation.

xy-goals, Zone-goals: The high and low-level policies are the same as before with the following minor differences. In xy-goals, the high-level policy outputs xy-coordinates which the low-level policy conditions on. In Zone-goals, the high-level policy outputs a discrete zone, and the low-level policy conditions on the xy-coordinates of that zone.

C.2. Hyperparameters

Hyperparameter tuning in RL is known to be laborious due to the large number of tunable hyperparameters, yet critical for performance. To ensure robust training, we set relatively

[2] https://github.com/openai/safety-gym
[3] https://github.com/lcswillems/torch-ac
[4] https://www.roboti.us/index.html

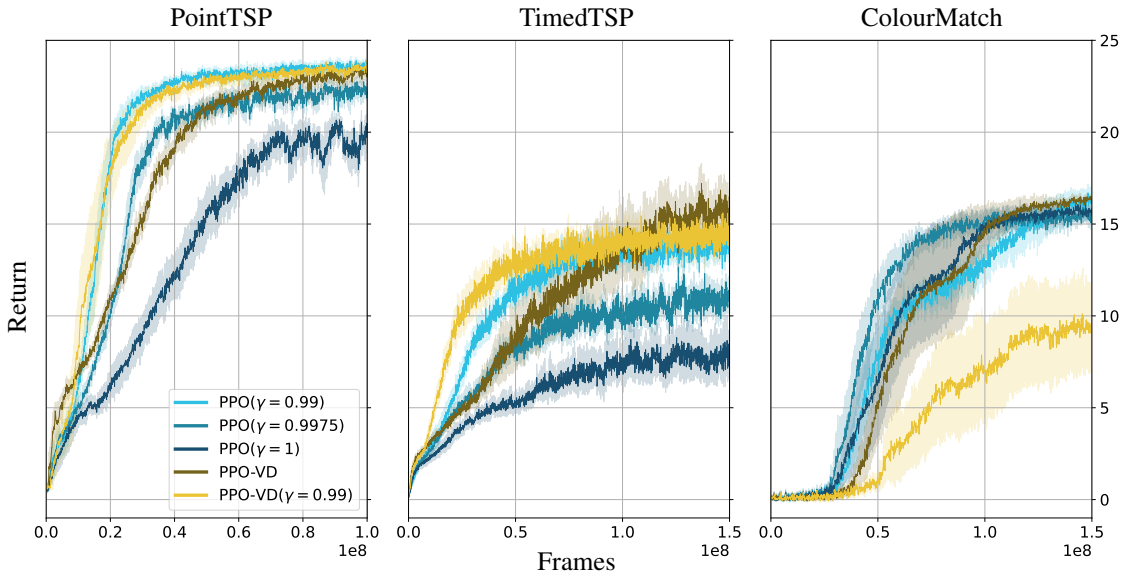


Figure 7. Learning curves for PPO and PPO$_{\mathcal{VD}}$ baselines. PPO$_{\mathcal{VD}}^{(\gamma=1)}$ generally results in similar or better final performance compared to PPO, but PPO$^{(\gamma=0.99)}$ can be more sample efficient. Results are averaged over 5 runs, with error bars reporting standard error.

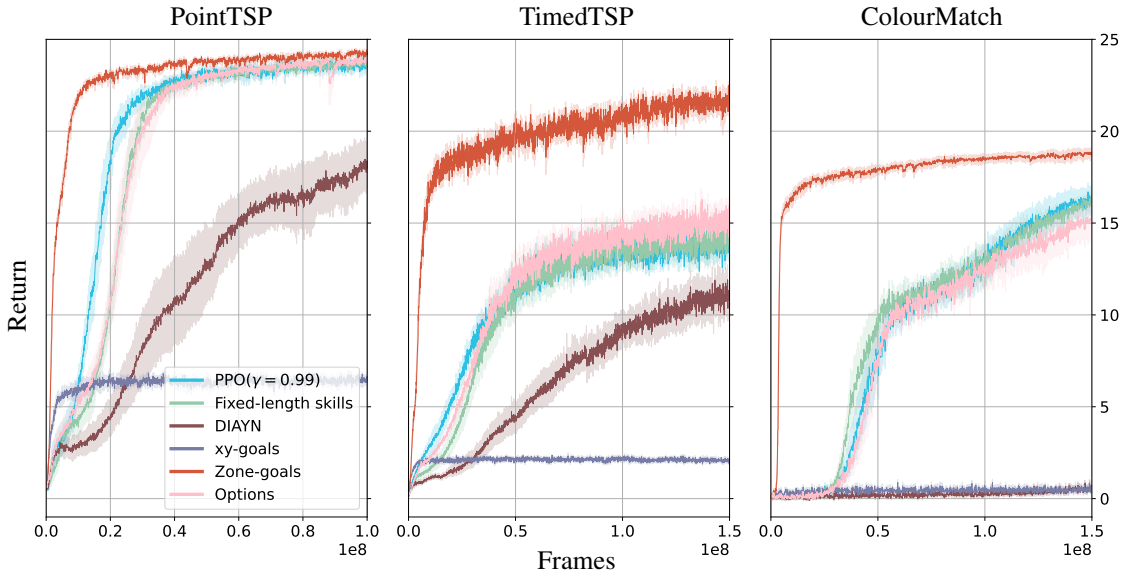


Figure 8. Learning curves for HRL baselines, with PPO$^{(\gamma=0.99)}$ included for comparison. Zone-goals significantly outperformed other baselines, while other HRL approaches performed similarly or worse compared to PPO. Results are averaged over 5 runs, with error bars reporting standard error.

Table 2. Hyperparameters for PPO and PPO$_{\mathcal{VD}}$ for the majority of our experiments. The following alterations were made: on ColourMatch, the number of steps per update was doubled to 128000 (for all methods); on PointTSP, PPO$_{\mathcal{VD}}^{(\gamma=1)}$ used 6 epochs for optimization.

	PPO			PPO$_{\mathcal{VD}}$	
Discount factor (γ)	0.99	0.9975	1	0.99	1
Value loss coefficient	0.5	0.5	0.5	0.005	0.005
Env. steps per update	$\leftarrow 64,000 \rightarrow$				
Number of epochs	$\leftarrow 10 \rightarrow$				
Minibatch size	$\leftarrow 1,600 \rightarrow$				
Learning rate	$\leftarrow 3 \times 10^{-4} \rightarrow$				
GAE-λ	$\leftarrow 0.95 \rightarrow$				
Entropy coefficient	$\leftarrow 0.003 \rightarrow$				
Gradient Clipping	$\leftarrow 0.5 \rightarrow$				
PPO Clipping (ε)	$\leftarrow 0.2 \rightarrow$				
Hidden units per layer	$\leftarrow 185 \rightarrow$				

large values for the number of env. steps per update and the minibatch size for all approaches while tuning the number of optimization epochs. The entropy coefficient was critical for final performance – setting the value too high would indirectly reduce the agent's maximum speed while setting the value too low would lead to suboptimal convergence. Since many of our baselines are based on PPO, we found a similar set of hyperparameters to perform well across many methods.

We report the hyperparameters for PPO and PPO$_{\mathcal{VD}}$ in Table 2 and the hyperparameters for the HRL methods in Table 3. For HRL methods with a fixed skill length (fixed-length skills, DIAYN, xy-goals), we chose a skill length of 200. For methods with a fixed number of discrete skills (fixed-length skills, DIAYN, Options) we set the number of skills to 5.

DIAYN also used a hyperparameter α to linearly combine environment rewards with the diversity objective, which was set to 0.01. We found that lower values of α caused the diversity objective to be ignored, resulting in similar performance to fixed-length skills.

D. Broader Impact

While our contributions are primarily foundational and may not directly lead to malicious use, we acknowledge that future applications that exploit our research could have the potential for a negative societal impact. In particular, the insights gathered from our results may help train reinforcement learning agents with greater capabilities, and therefore, greater potential for harm when misused. As many applications of reinforcement learning are safety-critical, including autonomous driving, robotics, and healthcare, we caution against over-trusting reinforcement learning, particularly in situations where the system was not trained on the environment in which it is being deployed. More work is needed to develop robust safety guarantees for RL before it can be applied in many real-world settings.

Table 3. Hyperparameters for HRL baselines (Fixed-length skills, DIAYN, Options, xy-goals, Zone-goals, TSP-solver). On ColourMatch, the number of steps per update was doubled to 128000 (for all methods). The high-level hyperparameters do not apply to TSP-solver, which did not have a high-level policy.

Env. steps per update	64,000
Hidden units per layer	128
(Low-level policy optimization)	
Number of epochs	10
Minibatch size	1,600
Discount factor (γ)	0.99
Learning rate	3×10^{-4}
GAE-λ	0.95
Entropy coefficient	0.003
Value loss coefficient	0.5
Gradient Clipping	0.5
PPO Clipping (ε)	0.1
(High-level policy optimization)	
Number of epochs	5
Minibatch size	80
Discount factor (γ)	1
Learning rate	3×10^{-4}
GAE-λ	0.95
Entropy coefficient	0.01
Value loss coefficient	0.5
Gradient Clipping	0.5
PPO Clipping (ε)	0.1