Neural Computing and Applications
https://doi.org/10.1007/s00521-024-09732-9

ORIGINAL ARTICLE

Hierarchical reinforcement Thompson composition

Güven Orkun Tanık[1] · Şeyda Ertekin[1,2]

Received: 10 January 2023 / Accepted: 25 March 2024

Abstract
Modern real-world control problems call for continuous control domains and robust, sample efficient and explainable control frameworks. We are presenting a framework for recursively composing control skills to solve compositional and progressively complex tasks. The framework promotes reuse of skills, and as a result quick adaptability to new tasks. The decision tree can be observed, providing insight into the agents' behavior. Furthermore, the skills can be transferred, modified or trained independently, which can simplify reward shaping and increase training speeds considerably. This paper is concerned with efficient composition of control algorithms using reinforcement learning and soft attention. Compositional and temporal abstraction is the key to improving learning and planning in reinforcement learning. Our Thompson sampling inspired soft-attention model is demonstrated to efficiently solve the composition problem.

Keywords Reinforcement learning · Thompson sampling · Artificial intelligence agents · Soft attention

1 Introduction

Reinforcement learning (RL) is concerned with exploration and learning of the best action sequences in an environment to maximize cumulative reward. The prominent algorithmic hurdle of reinforcement learning is delayed feedback and reward attribution. The practical hurdle is difficulty of reward shaping, efficient learning and changing environments and eventually bridging the simulation to real-world gap. Models trained in simulations may not be able to achieve meaningful success in the real world, and the cost of learning in the real world may be unfeasible, either in the number of trials needed or the modes of failure being too costly. There is a vast spectrum of different reinforcement learning algorithms trying to address all these problems, as well as reducing programming effort and increasing practicality. To get our models to perform well in the real world, ideally, we need our models to be easy to specify, train and modify. Failure to specify robust and safe models, usually in the form of reward shaping, leads to many concrete problems, namely, undesired side effects, reward hacking, unsafe exploration and brittleness to distributional shift [1]. Human learning has remarkable qualities that would help us solve some of these problems, if we manage to mimic those qualities. We are attempting to improve training efficiency through a human inspired skill combination framework, that produces faster learning, and easier reward specification. This is especially useful in real-world applications where model training can be very costly, and rewards can be very hard to specify.

When we humans try to learn a new skill, we usually do not start from scratch. We usually utilize our previously learned fundamental movements and patterns to build our new skills [2]. Our skills are usually made up of compositions of smaller skills, embedded in the larger skill and adapted on the fly to the specific problem. Taking cooking as an example; cutting, mixing and stirring are not learned with each recipe. Cutting may be modified depending on the utensil used or the component that is being cut, but essentially it is the same skill across the board in a wide variety of cooking recipes. What we are proposing is the reinforcement learning equivalent of the underlying principle: a set of basic skills, which can be learned thoroughly and exhaustively, that will be combined to achieve ever

✉ Güven Orkun Tanık
orkun.tanik@metu.edu.tr

Şeyda Ertekin
sertekin@metu.edu.tr

[1] Computer Engineering Department, METU, 06800 Ankara, Turkey

[2] CAD/CAM & Robotics Center, METU, 06800 Ankara, Turkey

Published online: 20 April 2024

more complex tasks. This approach aims to build models using simple sub-models, which are by definition easier to specify and train, to alleviate aforementioned problems. We are proposing a hybrid approach, that aims to practically train models on a pre-decomposed problem, to efficiently learn and later be easy to modify, which would be more efficient than classical single-stage learners.

Single-stage learners, as in most deep reinforcement learning models, tend to lose their performance when the task they are trained on is modified. This also means that these models are sensitive to environmental changes which complicates the transition from simulation to real world. Especially for single-stage learners, robustification [3] is needed for dynamic environments or real-world deployment [4], where the environment contains a lot of small changes, where weather affects the sensors and perception of the agent.

Some of the previous work to address these problems has been on task decomposition [5, 6] and its efficient application [7, 8]. Bacon et al. [9] renewed interest leading to [10] and combination with other reinforcement learning methods [11, 12]. Lent [13] proposes a two-layer decomposition in reinforcement learning in random neural networks as a method to improve efficiency and performance. A similar work [14] also embraces a composition approach where a policy layer is shared between different skills. Our method, Hierarchical Reinforcement Thompson Composition (HRTC), proposes a novel composition method that efficiently explores and learns to combine sub-skills which allows separately training the skills and meta-policy. Keeping with the analogy, we are not trying to learn or discover the ideal decomposition. The ideal decomposition, or some decomposition, is assumed to be provided by the experts that are setting up the learning task.

This expert intuition, imparted to the model as the task decomposition, may be viewed as a foundation for the model. The expert intuition can be fed into the system using input rules as in [15], which need advice modeling and is more complex. Our approach has multiple advantages. It builds a symbolic base, similar to [16], which helps in solving complex problems, outperforming neural models as evidenced in NeurIPS 2021 Nethack challenge and in [16]. It also removes the need to retrain sub-models and decreases the training needed to solve posed problems significantly. Most importantly, this decomposition makes reward shaping much simpler, making training sub-models easier for the programmer and the model at the same time.

Training efficiency is a metric that is taken into account in [13, 15, 16]. This paper outlines the efficiency gains of the proposed hierarchical learning method, which uses a Thompson sampling inspired beta distributed soft-attention model to combine pre-trained sub-models against selected single-stage state-of-the-art models.

2 Background

We will briefly go over the methods we are using and drawing inspiration from in our proposal.

2.1 Actor–critic methods

Policy optimization methods are one of the fundamental reinforcement learning algorithm families that are about finding an approximating function that maps the agent's current state to its next action. Actor–critic methods [17] are on-policy temporal difference (TD) learning methods that use both a value function (critic) and a policy function (actor). Policy gradient methods are used to train these systems.

Our approach is a derivative of actor–critic models. The components and behavior can be summarized as follows:

- Actor: Selects actions based on the current state.
- Critic: Evaluates the chosen actions and estimates their values.
- Initialization: Initialize actor and critic networks with random weights and set hyperparameters.
- Interacting with Environment: Actor selects an action, environment provides reward and next state.
- Updating the Critic: Critic's weights are updated to minimize the difference between predicted and actual rewards (TD error).
- Updating the Actor: Actor's weights are updated to maximize actions leading to higher rewards, guided by the critic's evaluation
- Repeat: Update steps are repeated for multiple episodes, enabling the model to learn optimal actions over time.

Vanilla policy gradient keeps the policy updates close in parameter space. However, small changes to some parameters may have huge effects on the model outputs with corresponding performance effects. Thus, small updates may end up collapsing the model performance. This makes the models brittle when using large step sizes with vanilla policy gradients, hurting policy gradients' sample efficiency.

In order to counteract the high variance that policy-based methods suffer from, it is possible to employ a base value function to de-bias the state-action expected value function. If high variance is not addressed, it may lead to catastrophic collapse of the learned function, or poor convergence.

Trust region policy optimization (TRPO) [18] and proximal policy optimization (PPO) [19] techniques build on these concepts. TRPO aims to update its policy by taking the largest step possible to improve performance, while satisfying a closeness constraint on the updated and old policies to avoid taking a too large update step that may

destabilize model performance. The constraint modeled using KL-Divergence, as a measure of distance between probability distributions. TRPO tries to avoid this kind of collapse using its closeness constraint and aims to quickly and monotonically improve performance.

PPO is another step in the same direction as TRPO. PPO also tries to take the largest "safe" policy gradient step, without jeopardizing model collapse. The main difference is the approach to the problem. TRPO tries to solve this problem with a complex second-order computation at each step, whereas PPO distills the closeness objective into its objective function. PPO is significantly simpler to implement, and even though the computations are not as precise, the resulting performance is on par with TRPO with much less computational overhead.

We are using a actor–critic method, specifically PPO in the core of our model. PPO is chosen because when comparing end-to-end methods, it outperformed all other methods we have tried in our experiments in terms of learning performance, namely, TRPO, Advantage Actor–Critic (A2C) [20], Deep Deterministic Policy Gradient (DDPG) [21], Soft Actor–Critic (SAC) [22] and Twin Delayed DDPG (TD3) [23].

2.2 Soft attention

We humans use our selective attention in our cognitive faculties [24]. We restrict our attention to particular objects and tasks and tune out irrelevant information. This helps us concentrate, and emphasize what is considered more important, urgent and relevant. In line with the parallels, we draw between human learning and our proposal, we are making use of attention systems for learning.

Computational attention mechanisms have been invented and have been successfully applied to deep learning problems, commencing a wave of performance gains in every problem domain that deep learning techniques are applied. Selective attention assumes not every input is of the same importance at a given step. This may be objects or pixels in image processing tasks, words and expressions in language processing or sub-tasks in our application. Attention mechanisms have two distinct categories: hard and soft attention [25].

Hard attention refers to choosing some part of the input and only processing that part. Whereas soft attention still takes in the whole input, but multiplies the whole input by some attention coefficient such that the relevant part is more prominent in its representation. This lends itself better to back-propagation and thus usually easier to train.

We are using soft attention in our model's action filter. This allows us to modify the resulting overall output action, combining all the candidate actions in a way that is driven by the current state.

2.3 Beta distribution

Deep learning-based models working in continuous environments usually output probability distributions which is then sampled to determine the resulting action. Classical algorithms use normal distribution as their output distribution. The model outputs a mean value and a standard deviation, which are used to build the probability distribution function.

However, there are drawbacks to this approach. A normal distribution is a symmetric distribution around the mean. When the mean moves to either end of the probability spectrum, the trailing end of the distribution corresponds to the nearest extreme end of the distribution results in sampling zero or one disproportionately more probable. As the mean of the distribution gets closer to zero, the probability of sampling zero increases disproportionately. This sampling "bias" results in models that converge and behave in sub-optimal manners as the numerical mean of the samples and outputted mean from the model start to diverge as the mean gets closer to zero or one.

However, beta distribution does not suffer from the imbalance that occurs when the distribution is close to zero or one. Beta distribution is approximately normal when its parameters are sufficiently large. Unlike normal distribution, which tracks the mean and variance of the distribution, beta distribution tracks the relative weights of its parameters, which results in a more natural relationship between successful application of the sub-tasks and their prevalence, which can be utilized as a pseudo-count.

Beta function can be defined as follows:

$$\beta(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)} \tag{1}$$

where Γ denotes the gamma function. The beta function's probability density function can assume many shapes, from uniform (when $\beta = \alpha = 1$), to approximating normal distributions ($\alpha > 1 \& \beta > 1$).

We are using beta distribution in our policy outputs, especially in our HRTC meta-policy but also by modifying the PPO algorithm to demonstrate the concept.

2.4 Thompson sampling

Thompson sampling is an algorithm that is used to optimize the explore–exploit dilemma in decision problems. It is a computationally efficient method that balances maximizing current performance and investing in exploration to potentially improving future performances. This technique is the critical component of our method, which allows us to improve the learning performance.

In ϵ-greedy approaches, as in a lot of the reinforcement learning algorithms, there is dithering for exploration. This

approach fails to write off bad actions, effectively wasting some of the exploration actions on experiments that have no possibility of improving the overall results. Dithering hurts results by delaying convergence, especially in large action spaces, like the target continuous environments of this work.

Thompson sampling samples a prior distribution for the decision, updating the expectations accordingly. If we assume the distribution to be beta, this leads to wider probability distributions where the actions are under-explored, and narrower probability distributions where the actions are better explored. As the decision process uses sampling, wider distributions have larger probabilities to be chosen if their expectation function can produce higher rewards.

Let's assume that there are k actions, and action k produces a reward with probability θ_k, and we have independent prior belief over each θ_k. If we take the prior to be beta distributed, prior probability density function for each action k becomes:

$$p(\theta_k) = \frac{\theta_k^{\alpha_k - 1}(1-\theta_k)^{\beta_k - 1}}{\beta(\alpha_k + \beta_k)} \tag{2}$$

where we can treat α and β as pseudo-counts, with positive rewards increasing α and negative rewards increasing β. With this simple update scheme, a stationary problem can be learned very efficiently. As $\alpha_k + \beta_k$ increases, the distribution becomes more concentrated. This means as an action is chosen over and over, the system prior belief about the outcome for that action becomes more precise.

On the other hand, when an action is under-explored, the resulting sampled value can have a wide range of values, which makes it more likely to be chosen to be explored, which, in turn, concentrates the distribution function, resulting in efficient exploration. For the exploitation phase, simply the mean for the beta function $\alpha_k/(\alpha_k + \beta_k)$ is used.

We are making use of a Thompson sampling inspired action filter, which essentially decides the more desirable the contribution of each sub-action at each state. This mechanism feeds into the combination logic directly, providing a simple and easy to compute combinational action at each state, while providing an efficient way for exploration by minimizing dithering. Classical explore–exploit approaches can be considered to use the principle of optimism in the face of uncertainty (OFU). Our approach, like [26, 27] in Bayesian systems, applies plausibility into exploration instead of optimism.

2.5 Hierarchical reinforcement learning

As our method uses a decomposition in its core, we need to mention hierarchical reinforcement learning methods, even though unlike most of these methods we are not trying to find an ideal decomposition of tasks, these methods gave us inspiration on the formulation of our method. We are assuming the decomposition is supplied to us. Essentially imposing a decomposition task on the programmer, is a performance advantage in the shape of not searching for the decomposition. Also simultaneously changing decomposition and combinations increase the complexity, delaying the learning process.

All the previously mentioned methods are end-to-end methods, which take a problem, and try to solve it at once. Then, there are the family of hierarchical learning methods for decomposing the problem and solving each sub-problem.

Feudal Networks [6] learn to divide the work to achieve goals at each level, so that the manager level learns to assign local and specific sub-goals to the lower level, while the lower levels learn to solve their tasks optimally.

Options Framework [5], specifically option–critic architecture [9, 11, 12], attempts to learn a decomposition termed options, with an actor–critic model to choose the option to be executed. Options are executed until their termination, then a new option is selected via an initiation function. The framework learns when to terminate and initiate an option.

These methods introduce implicit temporal abstractions as a complexity management method. Our method uses explicit task-based temporal abstraction.

3 Methods

Our proposed algorithm aims to build a skill combination model. In order to achieve this, we are combining a Thompson sampling like approach with skill decomposition and show that both are needed for a positive outcome. As illustrated in Fig. 1, there is a meta-policy and a master critic. Master critic is the value function for the whole combined skill. However, the action is not produced by the meta-policy itself. The meta-policy is there to learn how to combine the outputs of the various sub-skills using soft attention. Meta-policy is learned in an on-policy manner using PPO with a modification—beta distribution.

Considering the general case as a finite horizon Markov decision process with $M = (S, A, R, P, \tau)$, where S is the state space, A is the action space, $R_{\mathrm{a}}(s)$ is the probabilistic reward function for action a in state s, $P_{\mathrm{a}}(s'|s)$ is the probability distribution for transitioning to state s' from state s when action a is chosen and τ is the time horizon. A deterministic policy μ is a mapping from each state $s \in S$ to an action $a \in A$. We define the value function as

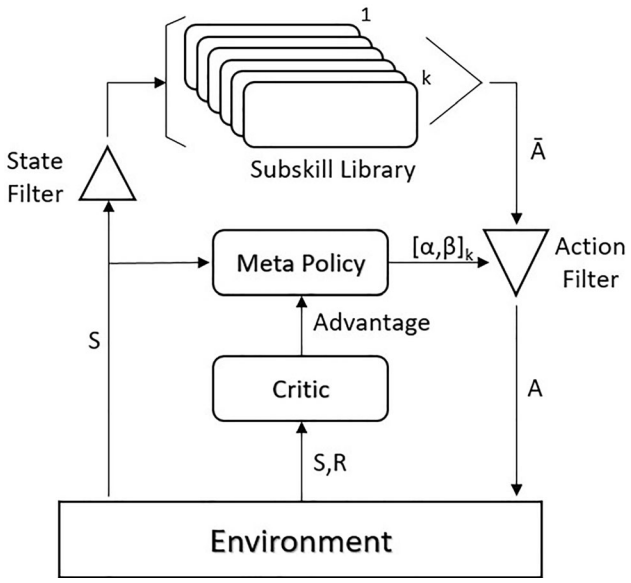

Fig. 1 HRTC algorithm

$$V_{\mu,i}(s) = E_{\mu} \sum_{j=i}^{\tau} R_{\mathrm{a_j}}(s_j) \tag{3}$$

where $R_{\mathrm{a}}(s)$ denotes the expected reward when the action a is selected in state s. The policy μ is optimal for M when $V_{\mu,i}(s)$ is maximal for $\forall s \in S$ and i. Our approach uses an actor–critic approach, where V is approximated by an neural network called the critic, and meta-policy μ is learned by the critic neural network which tries to optimize itself according to the critic. Value function is a representation of the TD error and approximation of state-action values, stands for the anticipated reward for the agent following its policy μ. PPO loss function is used as is, which is clipped $r(\theta)\hat{A}_t$ for learning, is given in Eq. (4).

$$L_{\mathrm{PPO}}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{4}$$

where probability ratio r_t is defined as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)} \tag{5}$$

This loss equation attempts to regularize policy updates using the parameter ϵ, so that the updated policy is not far from the old policy. This mechanism intends to stop the update of the probability ratio after a threshold. PPO is an on-policy approach that trains a stochastic policy, which does exploration by sampling actions according to its latest policy. This work is an empirical attempt to modify the actor for efficient exploration with powerful generalization using Thompson sampling, while avoiding dithering and using sub-skills to achieve exploration coverage of the action space.

Actor–critic methods usually produce σ and μ values defining a normal distribution, corresponding to the mean and variance, respectively, of said distribution. The agent then samples this distribution as a mechanism for exploration, updating σ and μ values in the process. During performance evaluation, usually directly σ is used as the output value instead of sampling, leading to a more stable output. In order to embed the Thompson sampling mechanism, we are modifying the actor to output α and β values for each sub-skill in the library for each context, then sampling the resulting beta distribution as the input weights for a soft attention filter to combine the sub-skills. The beta mean $\alpha_k/(\alpha_k + \beta_k)$ is used during evaluation instead of sampling. This is akin to solving the learning problem with Thompson sampling in contextual bandit problems, albeit with delayed rewards, which is being taken care of using the critic network.

Filters are used to integrate different parts of the algorithm together seamlessly. State filter is used simplify sub-skills. Part of the simplification stems from using a subset of the observation because some sub-skills may not make use of the whole observed environment. The other part of simplification is simplification of the desired result, as it is just a small part of the problem. At the output end, all the sub-skills simultaneously run as if the context entails the full usage of each sub-skill. Each sub-skill produces their proposed actions their own reward systems would entail. Action filter is then used to combine outputs of various sub-skills into a single action, using soft attention.

At each step, all the sub-skills read in their filtered state and produce their intended output. Some of the skills will be antagonistic, in which case the meta-policy quickly learns to use one of the antagonistic sub-skills. Some of the skills are synergistic, which are regulated and used in combination by the meta-policy.

Algorithm 1 HRTC, single level

procedure HRTC
  for iteration=1,2,... **do**
    for actor=1,2,...,N **do**
      for T Timesteps **do**
        Get State S, Reward R;
        Compute Advantage Estimate $\hat{A}$;
        Compute Action candidates
        $\bar{A} = SubSkill(StateFilter(S))$;
        Compute $A = \beta(\pi_{\theta_{master}}) \cdot \bar{A}$;
        Take environment step using A
      end for
    end for
    Optimize surrogate L wrt θ , with K epochs and minibatch size $M \leq NT$;
    Update θ_{master};
  end for
end procedure

This novel approach has several advantages:

- Each sub-skill can be a different type of model;
- Optimized/chosen for the skill needed, such as MPC for navigation
- Can be individually trained
- Can have its own state subspace
- Have its own reward function
- Can be individually verified/tested
- Can be trained without affecting other sub-skills
- Meta-policy can be thought as a skill itself, recursively building more complex skill trees
- Meta-policy and skills can have distinct time resolutions
- Environment effects can be compensated in the meta-policy, which drastically simplifies skill training
- Skills can be transferred
- Usage and activation of the skills can be observed and modified, which combined with pre-decomposition of skills, lends itself to being explainable which is extremely important in real-world systems.

Our results demonstrate that this approach converges faster and more reliably in reinforcement learning tasks under control domain.

4 Results

When we set out to find a suitable demonstration environment for our proposed model, we were looking for a fairly simple continuous control environment that had random elements. The environment's randomization is crucial for our principal value proposition, an adaptable algorithm.

We chose our demonstration medium as OpenAI Gym's CarRacing-v0 environment. It is a fairly simple continuous control environment that incorporates a randomly generated track, and a car which has acceleration, braking and steering as inputs. This environment was chosen for the initial evaluation and demonstration purposes, as it offers a simplistic state and action space. Reward structure is also fairly simple: There is a continuous time penalty of $-0,1$ points per frame designed to encourage forward motion, and a positive $+1000$/N points reward for any new section of the track that the car moves through, where N is the total number of tiles in the track. If the agent achieves 900 points average in 10 consecutive tracks, we consider the problem solved. If the problem is not solved in 10,000 tracks, we consider the model failed to solve the problem.

The reward structure encourages the controller to quickly complete the track, and the random track generation coupled with our 10 consecutive track rule punishes memorization.

We also opted for a static sub-skill library that does not require training to clearly demonstrate the performance of our algorithm. For implementation, it is assumed as deterministic action outcomes for five distinct action intents: turn left [−1., 0., 0.], turn right [0., 0., 0.], accelerate [0., 1., 0.], decelerate [0., 0., 0.8] and hold condition [0., 0., 0.]. Meta-policy learns to combine these intents (actions), as its response to observed states. We intentionally chose a large number of sub-skills intentionally to demonstrate combination power of our algorithm, instead of a more efficient already combined two sub-skill setup; just a turning and an accelerate/decelerate skill. Also we

chose the sub-skills such that they will need to be combined in antagonistic and synergistic manners. In our setup, acceleration–deceleration and right–left turns are antagonistic pairs as both produce opposite results. Whereas acceleration and either turns are synergistic that in order to navigate both should be employed at the same time. For example, to successfully execute a turn around a bend, the model should learn to decrease acceleration and increase deceleration prior to entering the bend, then accelerate and turn in the correct direction while navigating the turn.

Our proposed model solved the environment faster and more reliably than other methods. The options based methods failed to converge in 10,000 steps in any of the trials. However, single-stage methods were competitive. We benchmarked PPO, SAC, TD3 and A2C—a combination of on-policy and off-policy algorithms with a mix of stochastic and deterministic policies. PPO is used for benchmarking, as it is the best performing in experiments, and HRTC uses a derivative of PPO for its meta-policy learner. PPO algorithm itself is also modified with beta distribution to demonstrate the difference is not only due to an output distribution change, which coincidentally performed among the worst in the experiments. Also almost all the remaining code, and the actor and critic networks are sized identically to show the advantages of our method. We used a 6-layer convolutional network, taking in a $4 \times 96 \times 96$ input, and outputting $256 \times 1 \times 1$. A 2-layer fully connected actor and critic networks were connected to complete the system. Same networks' outputs were modified to output α and β values for beta distribution modifications.

As it is observed from Table 1, our proposed method achieves better convergence characteristics overall. Training is much more stable, and variance of the results are much smaller. The continuous variants of the environments we chose come with relatively small action spaces, even then a significant number of the training attempts fail. We opted to use sequential seeds in order to capture the stability aspect of the algorithms. Results where PPO model collapsed are omitted from the result in order not the negatively bias the results of the collapsing models, number of collapses are noted at the bottom of the table. The number of runs denoted on the table also directly correlate with the running time of the respectable algorithms, as all four models use the same underlying structure. Each run of the CarRacing-v0 environment lasts 1000 frames, thus the number of runs directly equate time spent in the environment.

As it is observed in Fig. 2, the algorithm performance converges faster and with smaller fluctuations, as can be deduced from the smaller variance figure of our algorithm in Table 1. Visual inspection of the trained models in the environment also showed good error correcting behavior from the HRTC agent, which found road tiles quickly and resumed driving as fast as possible.

We also ported our solution to continous variant of LunarLander gym environment, using a static heuristic sub-routine library as the HRTC sub-skill set. This environment is a classic rocket trajectory control task, where the agent tries to land a rocket ship softly to a landing pad, using the main engine and two orientation engines. Agent loses points for firing engines and gets rewards for slowly approaching the landing pad. Results are shown in Fig. 3. This result shows that using a well performing sub-skill set, the algorithm convergence characteristics can be reproduced in other environments.

Table 1 Performance—Numbers denote the number of runs until solution

Trial number	PPO		HRTC		SAC	TD3	A2C
	Beta	Normal	Beta	Normal			
0	3620	3300	**1890**	2150	3120	1920	2830
1	**1570**	2460	1680	1990	3430	4580	2670
4	1670	**1650**	1760	2950	6490	2560	2740
5	**1670**	1750	1850	2380	2220	2790	2100
6	5140	2490	**2150**	2470	2760	2630	2880
7	3080	2820	**1860**	2240	3510	2150	2320
8	5200	1840	**1590**	3170	2680	2280	3060
9	2160	**1920**	2350	2650	7440	3000	1940
10	2120	3710	**1290**	2250	2790	1870	2550
Model collapse	2	1	**0**	1	3	1	1
Fastest solution	2	2	**5**	0	0	0	0
Avg± STD Dev	2914± 1449	2438± 727	**1824± 307**	2472± 386	3827± 1733	2642± 775	2566± 353

Bold values indicate the best performance

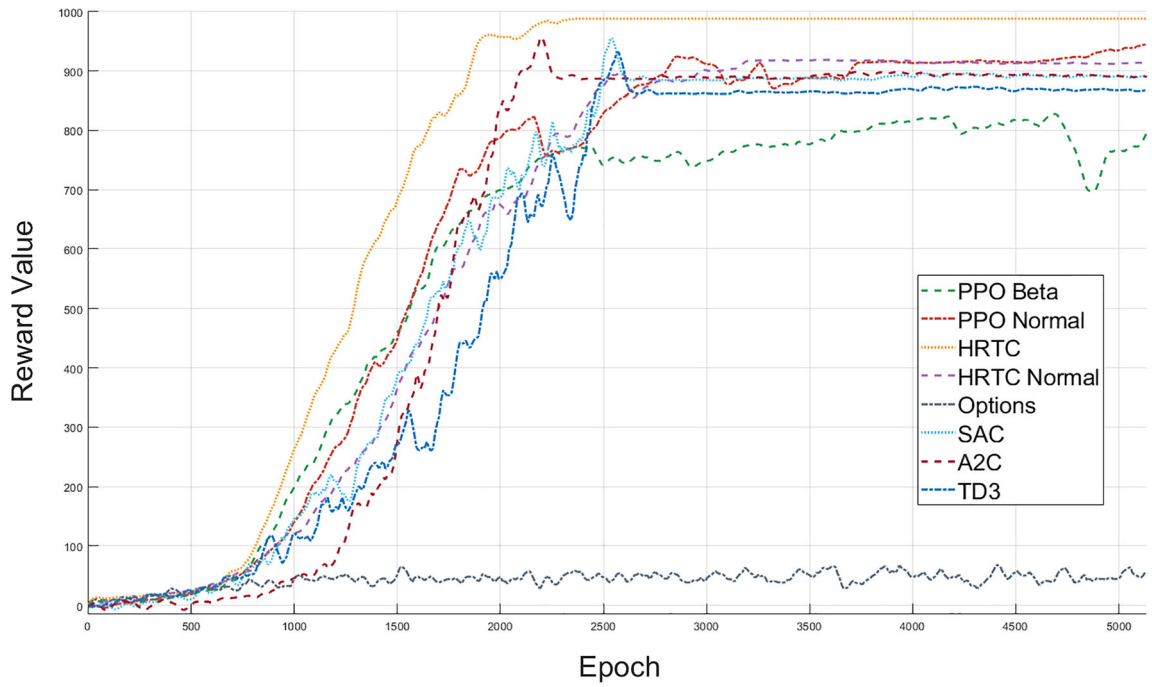

Fig. 2 Algorithm training results. *X*-axis depicts the number of epochs. *Y*-axis is the total reward running average accumulated and smoothed for visual clarity. The trends are unchanged until the end of the experiment at epoch 10,000, which is cropped in favor of the observability of solution performance

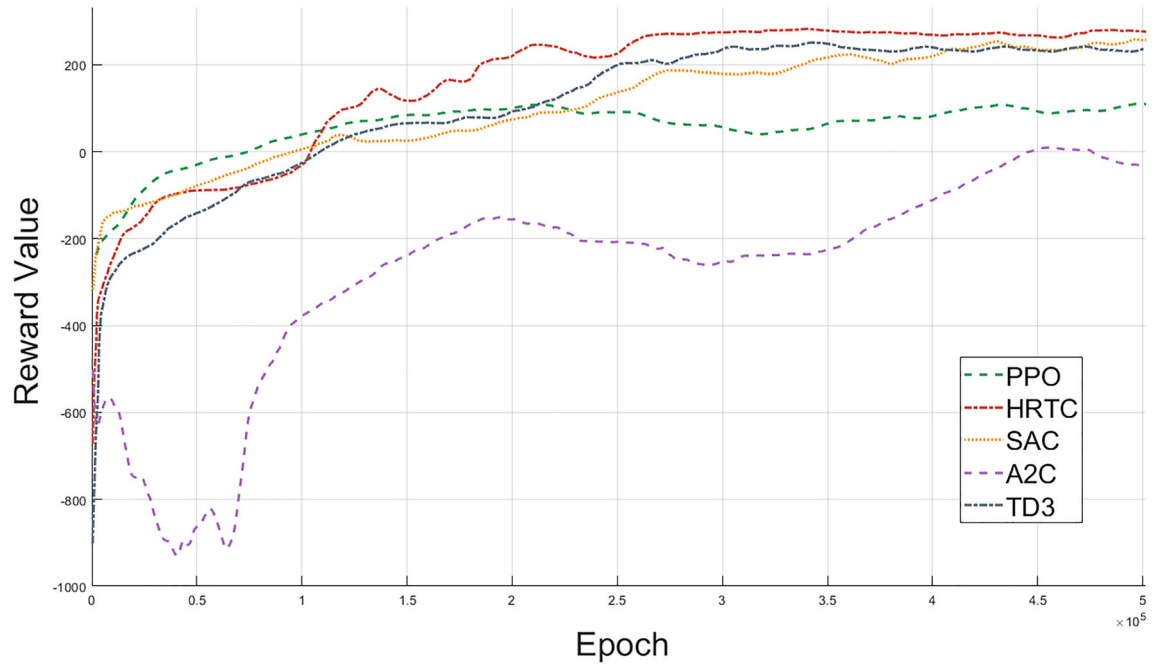

Fig. 3 Algorithm training results on LunarLander environment. *X*-axis depicts the number of epochs. *Y*-axis is the total reward running average accumulated and smoothed for visual clarity

The main difficulty of analyzing the results is detangling the interaction of the beta distribution—Thompson sampling and the meta-policy learning, as the results demonstrate that the HRTC normal also shows some satisfactory results. Table 1 also shows that the HRTC variants have less result variance, which may be due to the

effect of the static intents. We attempted to show the attribution of the results between the two modifications we made, distribution and meta-policy, via experimental results.

5 Conclusions

We demonstrated a novel approach to hierarchical reinforcement learning that is efficient, stable, explainable and versatile that can be used in control settings. It is especially useful where incremental changes to the environment, model or mission are expected.

The demonstrated results show better efficiency in learning, as the compared methods all make use of on-policy learners with no memory replay. Even when all the sub-components are the same, the Thompson sampling system, with its prior beliefs and pseudo-counts learned through experimentation, achieves better and more efficient characteristics than some of the most efficient and most widely used state-of-the-art RL methods. Our proposed method even had a slight disadvantage in the number of parameters to learn.

This work will presumably have more impact in more complex environments, as said environments would require much higher learning costs for each required sub-skill. Future experiments in more complex environments, such as in high-fidelity simulators, will be conducted to confirm our assertions and quantify associated efficiency gains. Specifying a good reward function can be arduous and prone to error, as the agents may not behave as expected. The potential impact of this framework would also make reward functions compositional, thus making specifying reward functions easier since they will only be concerned about the sub-skill that they are specified on. Reuse and partial trainability will also make the whole training more efficient.

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK). No funds, grants or other support was received.

Data availability The data sets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

1. Amodei D, Olah C, Steinhardt J, Christiano P, Schulman J, Mané D (2016) Concrete problems in AI safety
2. Hangl S, Dunjko V, Briegel HJ, Piater J (2020) Skill learning by autonomous robotic playing using active learning and exploratory behavior composition. Front Roboti AI. https://doi.org/10.3389/frobt.2020.00042
3. Cheng Y, Zhao P, Wang F, Block DJ, Hovakimyan N (2022) Improving the robustness of reinforcement learning policies with l1adaptive control. IEEE Robot Autom Lett 7:6574–6581. https://doi.org/10.1109/LRA.2022.3169309
4. Amini A, Gilitschenski I, Phillips J, Moseyko J, Banerjee R, Karaman S, Rus D (2020) Learning robust control policies for end-to-end autonomous driving from data-driven simulation. IEEE Robot Autom Lett 5:1143–1150. https://doi.org/10.1109/LRA.2020.2966414
5. Sutton RS, Precup D, Singh S (1999) Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. Artif Intell 112:181–211. https://doi.org/10.1016/S0004-3702(99)00052-1
6. Vezhnevets AS, Osindero S, Schaul T, Heess N, Jaderberg M, Silver D, Kavukcuoglu K (2017) Feudal networks for hierarchical reinforcement learning
7. Frans K, Ho J, Chen X, Abbeel P, Schulman J (2017) Meta learning shared hierarchies
8. Nachum O, Gu S, Lee H, Levine S (2018) Data-efficient hierarchical reinforcement learning
9. Bacon P-L, Harb J, Precup D (2016) The option-critic architecture
10. Riemer M, Liu M, Tesauro G (2018) Learning abstract options
11. Chunduru R, Precup D (2020) Attention option-critic
12. Kamat A, Precup D (2020) Diversity-enriched option-critic
13. Lent R (2019) A generalized reinforcement learning scheme for random neural networks. Neural Comput Appl 31:2699–2716. https://doi.org/10.1007/s00521-017-3223-1
14. Sahni H, Kumar S, Tejani F, Isbell C (2017) Learning to compose skills
15. Bignold A, Cruz F, Dazeley R, Vamplew P, Foale C (2021) Persistent rule-based interactive reinforcement learning. Neural Comput Appl. https://doi.org/10.1007/s00521-021-06466-w
16. Kurniawan B, Vamplew P, Papasimeon M, Dazeley R, Foale C (2022) Discrete-to-deep reinforcement learning methods. Neural Comput Appl 34:1713–1733. https://doi.org/10.1007/s00521-021-06270-6
17. Konda VR, Tsitsiklis JN (2000) Actor-critic algorithms
18. Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P (2015) Trust region policy optimization
19. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms
20. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap TP, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning

21. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning
22. Haarnoja T, Zhou A, Hartikainen K, Tucker G, Ha S, Tan J, Kumar V, Zhu H, Gupta A, Abbeel P, Levine S (2018) Soft actor-critic algorithms and applications
23. Fujimoto S, Hoof HV, Meger D (2018) Addressing function approximation error in actor-critic methods. https://github.com/
24. Bater LR, Jordan SS (2019) Selective attention. Springer, Berlin, pp 1–4. https://doi.org/10.1007/978-3-319-28099-8_1904-1
25. Niu Z, Zhong G, Yu H (2021) A review on the attention mechanism of deep learning. Neurocomputing 452:48–62. https://doi.org/10.1016/j.neucom.2021.03.091
26. Osband I, Russo D, Roy BV (2013) (More) efficient reinforcement learning via posterior sampling
27. Osban I, Roy BV (2016) Why is posterior sampling better than optimism for reinforcement learning?