2018 7th Brazilian Conference on Intelligent Systems

A Framework to Discover and Reuse Object-Oriented Options in Reinforcement Learning

Rodrigo Cesar Bonini[1], Felipe Leno Da Silva[1,2], Ruben Glatt[1], Edison Spina[1], Anna Helena Reali Costa[1]
[1]Universidade de São Paulo, Brazil
[2]University of Texas at Austin, USA
{rodrigo_cesarb, f.leno, ruben.glatt, spina, anna.reali}@usp.br

***Abstract*—Reinforcement Learning is a successful yet slow technique to train autonomous agents. Option-based solutions can be used to accelerate learning and to transfer learned behaviors across tasks by encapsulating a partial policy. However, commonly these options are specific for a single task, do not take in account similar features between tasks and may not correspond exactly to an optimal behavior when transferred to another task. Therefore, unprincipled transfer might provide bad options to the agent, hampering the learning process. We here propose a way to discover and reuse learned object-oriented options in a probabilistic way in order to enable better actuation choices to the agent in multiple different tasks. Our experimental evaluation show that our proposal is able to learn and successfully reuse options across different tasks.**

***Index Terms*—reinforcement learning, transfer learning, object-oriented options**

I. Introduction

Reinforcement Learning (RL) [1] allows agents to learn autonomously through interactions with an environment. An action that affects the environment is chosen by the agent, then the agent observes how much that action helped to the task completion through a reward function. An agent can learn how to optimally solve tasks by executing this process multiple times. Although RL has been successfully applied in many problems [2]–[4], classical algorithms require a huge number of samples for learning a task.

Transfer Learning (TL) [5] and the Options framework [6] are examples of efforts to accelerate learning. While the former relies on reuse of previous knowledge, the latter offers a way to encapsulate sequences of actions in a high-level behavior to be performed by agents. Combining those two ideas is a promising way to accelerate learning. However, transferring learned behaviors across tasks is not trivial, as the agent must cope with differences between tasks.

Some authors have attempted to use options for TL by discovering them with a focus on subgoals or through reference points to guide agents to learn a task faster [7], [8]. However, in domains and tasks where there are no clear subgoals or reference points, it is preferable to merge multiple parts of policies instead, as an attempt to make the behavior less overfitted to a task and easier to be applied later [9], [10].

While in early proposals [11], [12] options were only manually built or mapped by domain experts, recent proposals are able to autonomously discover options and transfer them across different problems [13]–[15]. However, those works assume the tasks to be very similar, resulting in negative transfer if a previously learned behavior is not directly applicable in the new task where learning is not accelerated but hurt instead through disadvantageous knowledge. The work proposed in [13] does not have a good initiation set of the options and it may produce negative transfer if the tasks are too different and the agent follows a bad discovered option too long. The work proposed in [15], assumes that all options apply everywhere, no matter what it is, but not having a score method to evaluate the quality of the options before learning can hamper the learning process.

A possible way to alleviate negative transfer is to reuse multiple policies in new tasks [10], [11], [16], [17]. Fernandez and Veloso [16] probabilistically reuse a library of past deterministic policies that solve different tasks within the same domain. Even though alleviating negative transfer issues in simple domains, they reuse whole policies in the new task, while we here focus on reusing only partial policies (i.e., options) to transfer only the relevant behaviors. Moreover, they store a growing library of policies, which is not as effective as merging the policies into a single one as shown in [10]. The framework proposed by Koga *et al.* [10] combines multiple policies into a single stochastic one. In this way, it is possible to generalize the usually highly-specialized policies that are learned in a task, and better reuse them in a new task. However, their work focuses on combining whole policies, and they do not consider how to transfer only relevant behaviors.

Although the literature has shown that combining whole policies delivers a good performance for TL [10], [16], reusing options has been shown to be better than reusing whole policies in some situations [11]. If properly built, options might represent parts of the solution that are common across many related problems and therefore they can be used to transfer only the reusable part of the policy. However, the state-of-the-art approach for reusing options across tasks does not autonomously learn the options and needs a human for specifying them [11]. Furthermore, learning how to properly learn and reuse options between different tasks can be hard, since learned options are usually overfitted to a single task and hard to adapt to differences in the tasks [9], [14].

We here contribute with a framework, hereafter called PRDO (Probabilistic Reuse of Discovery Options), to: *learn* options that encapsulate behaviors in a task; *combine* them into a single probabilistic option that encapsulated the learned

978-1-5386-8023-0/18/$31.00 ©2018 IEEE
DOI 10.1109/BRACIS.2018.00027

behaviors in a generalized manner; and *reuse* it in new tasks to accelerate learning.

We show in our experimental evaluation that PRDO adapts better to new tasks than the state-of-the-art option-transfer algorithm [13]. PRDO represents a new step towards autonomous agents that can autonomously reuse learned behaviors across similar yet different tasks [18].

II. Foundation

In this section, we first introduce the relevant basic concepts of RL, describe the extension to Object-Oriented MDPs, and present the concept of Options in more detail.

A. Reinforcement Learning

The RL framework [1] allows autonomous agents to learn through interactions in the environment. Many sequential decision problems are modeled by a Markov Decision Process (MDP) [1], and RL is an extensively used solution for MDPs driven by environment interactions. An MDP is described by the tuple $\langle S, A, T, R\rangle$, where S is the set of environment **states**, A is the set of available **actions**, T is the **transition function**, and R is the **reward function** (the agent does not know T and R). The goal of the agent in an MDP is to learn an optimal **policy** π^* that maps each state to the actions that lead to the highest expected cumulative sum of rewards over the lifetime of the agent.

As the output from the transition and reward functions cannot be predicted in learning problems, the MDP can be solved through interactions with the environment, which can be accomplished for instance by the *Q-Learning* algorithm [19]. *Q-Learning* iteratively learns a *Q-value*, that aims to estimate the cumulative discounted reward associated with each state-action pair: $Q : S \times A \rightarrow \mathbb{R}$. At each decision step, Q is updated following: $Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)]$.

Q-Learning eventually converges to the optimal Q function: $Q^*(s,a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$, and Q^* can be used to define an optimal policy as: $\pi^*(s) = \arg\max_a Q^*(s,a)$.

However, the standard Q-Learning and classical RL techniques may be inefficient in environments with large state spaces.

B. Object-Oriented MDPs

The use of task descriptions that allow abstraction can help to accelerate the learning process. The *Object-Oriented* representation [20] enables generalization and grouping opportunities through an intuitively-given task description. In *Object-Oriented MDPs* (OO-MDP) the state space is abstracted through the description of a set of classes $C = \{C_1, ..., C_c\}$, where each class C_i is composed of a set of *attributes* denoted as $Att(C_i) = \{C_i.b_1, ..., C_i.b_b\}$. Each attribute b_j has a *domain*, $Dom(C_i.b_j)$, specifying the set of values this attribute can assume. $K = \{k_1, ..., k_k\}$ is the set of objects that exist in a particular environment, where each object k_i is an instance of one class $C_i = C(k_i)$, so that k_i is described by the set of attributes from its class, $k_i :: Att(C(k_i))$. Then, the MDP state is now observed as the union of all object states $s = \cup_{k \in K} k.state$, where an object state is the set of values assumed by each of its attributes at a given time, $k_i.state = (\Pi_{b \in Att(C(k_i))} k_i.b)$. We will refer to $|K|$ as the OO-MDP *domain size*. The *status* of an object at any given time is the Cartesian product of the current values for each attribute of its class.

The object-oriented representation of an environment simplifies the representation of policies and transition models by abstracting similar states. If state spaces from different RL domains or tasks share a common set of objects, then knowledge can potentially be transferred between these domains [21], [22].

C. Options

The *Options Framework* [6] can be used to alleviate the RL scalability problem. Options extend the usual notion of actions, providing closed-loop partial policies for taking actions over a certain period of time. Options are high-level actions that aim at decomposing MDPs in subtasks and solving them, providing temporally extended courses of actions. Some examples of options may include a car passing in determined locations, a traveler going to a distant city or unlocking a door, in other words, tasks that require a certain number of primitive actions to achieve a desired subgoal in a task.

Formally, an option is a conditional sequence of primitive actions defined as a three-tuple [6], $< \pi, \mathcal{I}, \beta >$, consisting of: a **policy** that can be deterministic $\pi : S \rightarrow A$, or stochastic π: S x $A \rightarrow [0,1]$; a set of **initiation states** $\mathcal{I} \subseteq S$; and a **termination condition** $\beta : S \rightarrow [0,1]$. The initiation set $\mathcal{I}$ is the subset of the state space in which the option can be executed, i.e, an option is available in state s_t if $s_t \in \mathcal{I}$. When the option initiation condition is satisfied and the agent selects it, π is followed until a termination condition β is met. The termination condition β is a probability function over states that defines the likelihood of interrupting the execution of an option when in that state. When an option ends, the agent might select another option or a primitive action, until a goal state is reached. Alternatively, the agent may also interrupt the option after a predefined number of steps, which means that β outputs a probability of 1 after a predefined number of steps.

The Q-value of options might be estimated similarly as in the regular *Q-Learning* algorithm. Let $Q^*(s,o)$ be the expected return when starting in state s and following option o.The option value is updated by [8]:

$$Q(s_t, o_t) \leftarrow (1-\alpha)Q(s_t, o_t) + \alpha\left[r + \gamma^n \max_{o \in O_{s_{t+n}}} Q(s_{t+n}, o)\right], \quad (1)$$

where $r = r_{t+1} + \gamma r_{t+2} + ... + \gamma^{n-1} r_{t+n}$ and n is the size of the option o.

Therefore, when an option terminates, its expected value is updated in a Q-table that can be used normally to derive a policy.

Note that this definition does not specify how to learn an option by interacting with the environment, nor how to adapt it to reuse in a new task.

III. Discovering and Reusing Options

In this section, we describe two works that form the base to our framework. The first is an approach for object-oriented options-discovery, *Portable Options Discovery*, while the second is an efficient method for probabilistic options reuse, *Reuse Option* [11].

A. Portable Options Discovery

Despite the success of the early approaches for using options in a single domain [12], [23], automatically learning options without human supervision and mapping them between different tasks are still hard problems. The *Portable PolicyBlocks Algorithm* proposes a way to map between tasks and a score method to evaluate the option discovery before storing and reusing them [13]. The original *Policy Blocks* algorithm [9] needed to learn for a long time to discover options that are useful only in a single domain. For this reason, this method is not appropriate for transferring options across tasks with different state spaces.

This algorithm was extended to *Portable Policy Blocks* [13], which are specialized for learning transferable options. Given a set of task solutions (usually optimal), the algorithm finds commonalities between the solutions, building a set of options based on that. For generalizing the state-action space of the tasks, an object-oriented representation is used.

Given a set of policies (solutions) L for the source tasks $\psi \in \Psi$ and a number i of desired options, the *Portable PolicyBlocks Algorithm* can be seen as a 6 step process: (1) it generates a powerset of all possible subsets of L; (2) for each subset, computes the **great common generalization** (GCG), that is the maximal set of objects that are present in all source tasks; (3) merges all policies contained in each subset of tasks into abstract option candidates; (4) **scores** all option candidates according to their similarity to L; (5) **subtracts** the option candidates from the initial powerset in order to avoid redundancy in the next options to be discovered; and (6) adds the best option candidate c to the set of options O. This process repeats until the algorithm finds i options or if there are no more policies in L.

After the end of the algorithm, the i best rated abstract options will be returned.

The returned options can then be reused in a new task. For that, the options are added as a possible action in the environment. When the agent selects the option, the abstract policy will be executed by finding an abstract state that has the most objects in common as possible with the current state in the new task. The states are combined into the abstract state and the average of the sum of their Q-Value is applied to the abstract state. If no such state can be found, the option is terminated and the agent has to select a primitive action in the new task.

Even though enabling the reuse of learned options by transferring a number of options to the new task, *Portable PolicyBlocks* might even result in negative transfer and estimating the quality of each option might take a long time. The literature has shown that combining multiple policies into a single one can be better than storing a library of policies to be reused [10]. Inspired by this idea, we combine *Portable PolicyBlocks* with another approach [11] for merging all the discovered options into a single one and executing it for predetermined steps.

B. Reuse Option

An approach to probabilistically reuse options was proposed by Bernstein [11]. He follows the intuition that an agent's probability of behaving in a certain way should be proportional to how often that behavior has been successful in the past. Thus, when an option is being executed (reused), the probability of an action being chosen at a given state is related to the number of policies (or options) in which that action is chosen at that state. Given a set of stochastic policies $\pi_1, \pi_2, ..., \pi_m$, with π_M being the *mixed policy* formed by averaging all the policies, the probability of π_M choosing each action in a state s is defined by:

$$\pi_M(s,a) = \frac{1}{m}(\pi_1(s,a) + \pi_2(s,a) + ... + \pi_m(s,a)), \quad (2)$$

In their work, an option always terminates after a fixed number of steps. A **n-step complete reuse option** tuple, $< \pi_M, S, n >$, is built from the mixed policy π_M, where n is the time-limit, i.e, it executes π_M for exactly n time steps. Options can be initiated at any initial state, $\mathcal{I} = S$.

Even though Bernstein's proposal showed promising results [11] for TL, he does not provide a way to learn the options from the source tasks. We combine Bernstein's ideas with *Portable PolicyBlocks* as a way to autonomously *learn* the options and then transfer them, requiring significantly less human effort than in Bernstein's work.

IV. Probabilistic Reuse of Discovered Options

Despite the successful approaches described in the previous sections, none of them provide simultaneously the following: (1) an automatic options-discovery method in OO-MDP multiple tasks; (2) a combination of the learned options; (3) a probabilistic reuse of previous learned and combined options that controls exploitation and exploration.

Thus, in order to define a representation that joins object-oriented options-discovery and a probabilistic reuse of those options, we introduce a framework, hereafter called *Probabilistic Reuse of Discovered Options* (PRDO) to combine learned object-oriented options from different tasks and reuse them probabilistically providing a good initialization set of actions without taking away the possibility of the agent visiting states that are still unexplored.

The idea of PRDO is to **learn** options for each source task independently with an option-discovery algorithm; and

to **reuse** them applying the probabilistic ideas from [10], [11] in different target tasks.

The learned options are then intended to optimize the learning process in the target tasks, guiding the agent towards better trajectories, while also allowing it to explore other new states without hampering the learning process.

Our proposal is described in Algorithm 1. As input it requires, a set of source tasks Ψ_s, a number i of desired options to be discovered, a number of steps n that the options discovery will be executed in the target task and a target task ψ_t. Firstly it initializes a set of options O in step 1. Then, in steps 2-4, we execute an options-discovery algorithm (we used the *Portable Policyblocks* in our work, but any other algorithm can also be used here), which returns i options for each source task independently (including all them in O).

After that, the policies of the options discovered are combined in step 5 into a probabilistic policy π_o :

$$\pi_o(a|s) = \frac{|O(a|s)| + 1}{|O_s| + |A|}, \tag{3}$$

where $\pi_o(a|s)$ is the probability of action a being chosen by the single option $< \pi_o, S, n >$ in the state s, $|O(a|s)|$ is the number of options in O that select a in state s, and $|O_s|$ is the number of options that are defined for state s.

The options are combined in a way that π_o selects actions with a probability according to the number of times that they appear in the previous options, where the more the action appears, the higher its probability.

Then, the learning process is executed in steps 6-8 in the target task with the resulting option $< \pi_o, S, n >$ as another choice in the set of actions.

The learning uses a procedure similar to the one proposed in [11], in equation 2. The equation 3 differs from equation 2 because of the sum +1 in the numerator. With this sum, there is no possibility of any action having no probability of being chosen (and in the equation 2 this possibility exists). Thus, for the equation 3, in the case of the option does not have any action mapped for some state, the agent will have the same probability to choose between all the actions available, similar to the random option introduced in [11].

Algorithm 1 PRDO

Require: Set of source tasks Ψ_s, desired number of options i, number of steps for options n, and target task ψ_t

1: $O \leftarrow \emptyset$ //empty set of options
2: **for each** source task $\psi_s \in \Psi_s$ **do**
3: $O \leftarrow$ *Options-Discovery Algorithm* (ψ_t, i)
4: **end for**
5: $\pi_o \leftarrow combine(O)$
6: **for** H episodes **do**
7: run learning in target task ψ_t using $\{< \pi_o, S, n >\} \cup A$
8: **end for**
9: **return** π_t //solution of ψ_t

V. Experimental Evaluation

We perform experiments in two different domains: *GridWorld* and *TaxiWorld*, in order to verify the benefits provided by the options-discovery algorithm and the probabilistic reuse of them. The experiments were performed in 10 different source tasks and 1 target task, for each of 20 trials. All the 10 tasks differ from the other ones in terms of amount and position of walls, position of goal and initial position of the agent. The comparisons were made between 1-step options (n = 1) ; 5-step options (n = 5) ; n-step options (Portable PolicyBlocks - PPB) without applying the probabilistic reuse, where options were applied according to the discovery of the algorithm, having different size; and Q-Learning. We adopted the following parameters: α = 0.2, $\gamma = 0.9$ and $\epsilon = 0.1$.

A. GridWorld Domain

In the first experiment, we tested PRDO in a simple navigation domain. The agent starts in a random non-terminal state and must perform 10 source tasks (one task at a time) (see figure 1).

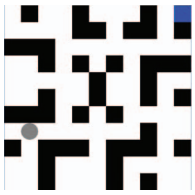

Fig. 1: Example of GridWorld task. The agent is represented by the gray circle, the goal by the blue square and the obstacles (walls) are in black.

Initially, the agent has to learn how to achieve the goal position as fast as possible independently in the 10 source tasks, where the options are discovered and stored.

For the learning of the initial policies with the options-discovery algorithm, we first performed the Q-learning algorithm for 100 episodes, providing 5 suboptimal policies to *Portable PolicyBlocks*, that extracts 3 options for each source task ψ_s. After that, those options in O are combined in $< \pi_o, S, n >$ using the equation 3 and evaluated in 1 different target task for each trial.

The action set available after learning the options is $A = \{north, south, east, west\} \cup < \pi_o, S, n >$. Episodes ends when the agent achieves the goal state, resulting in a reward of +1 discounted by γ = 0.9 and otherwise, the reward is 0 for any step. If moving to a wall, the agent stays in the same position.

In order to evaluate the relative effectiveness of the probabilistic learned options, we executed 100 learning episodes in the target task and took the mean steps to goal over the 20 trials.

Figure 2 shows the mean steps to goal of the 20 trials of the experiment in the Gridworld Domain, which has slightly similar tasks. The PRDO with 5-step options outperformed the Standard Q-Learning and PPB and 1-step options by learning faster at the beginning of the learning process. That happened

because 5-step options provide the agent a higher exploitation time than 1-step options, and not too much exploitation time like PPB. All 3 cases provide previous knowledge, which is better than only learning using Q-Learning.

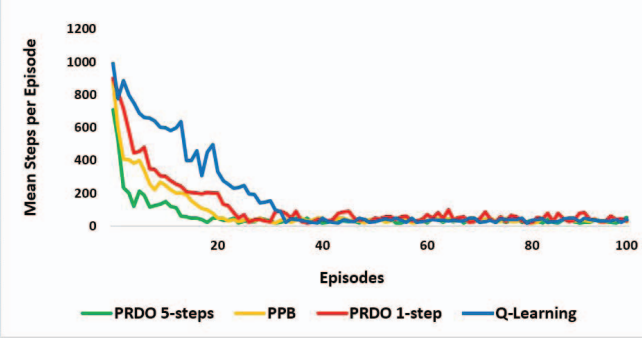

Fig. 2: The mean steps to goal for 100 episodes during the learning process over 20 executions in the target tasks of GridWorld Domain.

B. TaxiWorld Domain

In the second experiment, we tested PRDO in a more complex task. We included passengers to be picken up, to check if the agent would benefit from the stochasticity, being also useful when having to take them before going to the goal state and if it would perform better than PPB, which is an object-oriented options-discovery algorithm.

The agent (taxi) starts in a random non-terminal state and must perform 10 source tasks (one task at a time) with a different goal position to be reached in each of them, with passengers to pick up from different positions (see figure 3). The number of passengers available in the first evaluation (figure 4) source tasks is 4 and in the target task is 2, to check a scenario when learning in harder tasks and transferring to easier tasks. And the number of passengers available in the second evaluation (figure 5) in the source tasks is 2 and in the target task is 4, as suggested in [24], learning in easier tasks and transferring to harder tasks. These evaluations were done to check the symmetry of the framework, which works from larger tasks to a smaller task, and from smaller tasks to a larger one.

The action set available in these tasks after learning the options is $A = \{north, south, east, west, pick\ up, drop\ off\} \cup < \pi_o, S, n >$. The agent may *pick up* a passenger when in the same state, and may *drop* them *off* them in any state. Episodes ends only when the agent achieves the goal state after taking all the passengers, resulting in a reward of +1 discounted by γ = 0.9 and otherwise, the reward is 0 for any step. In case of moving to a wall, the agent stays in the same position.

Here, as the first experiment, the agent needs to achieve the goal position as fast as possible independently in the 10 source tasks, but taking all the passengers before getting to the goal.

In order to extract the options, we performed the Q-learning algorithm for 1000 episodes, providing 5 policies to *Portable PolicyBlocks* extracts 3 options for each source task ψ_s. After

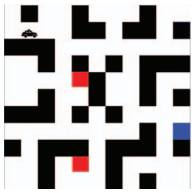

Fig. 3: The TaxiWorld Domain. The agent is represented by the taxi car, the goal by the blue square, the obstacles (walls) are in black and the passengers the agent may pick are in red. The agent must pick all the passengers before get to the goal.

that, those options in O are combined in π_o using the equation 3 and evaluated in 1 different target task.

We executed 10000 learning episodes for all cases in the target task and took the mean steps to goal over 20 executions. Due to the computational cost, we limited each episode to 10000 steps.

Figure 4 shows the mean steps to goal in 20 trials of the experiment in TaxiWorld Domain in the larger to smaller task and figure 5 shows the results in the smaller to larger task. In both cases, the PRDO framework with 1-step options was the best one, outperforming the Standard Q-Learning, PPB and 5-step options. That happened because 1-step options provide the agent more exploration time than 5-step, and PPB, which is better in a different and more difficult task. All of them again were better than Standard Q-Learning. This is important to see that exploitation is not always the best solution, because the passengers change their position and even in slightly similar tasks, when the complexity of the task is changed, these approaches (PPB, 5-step options) do not work very well if they do not have very good initial solutions and allow instant changes (updates) in their policies.

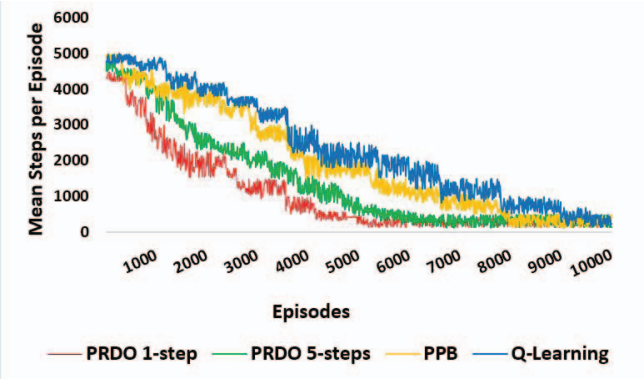

Fig. 4: The mean steps to goal for 10000 episodes during the learning process over 20 executions in the target tasks of TaxiWorld Domain. 4 passengers in the source tasks, 2 passengers in target task.

The Standard Q-Learning without options was only able to achieve similar results after almost 10000 learning episodes in the TaxiWorld Domain experiments. This difference between the mean steps to goal indicates that PRDO provides a speed-up in the learning process, greatly improving the performance

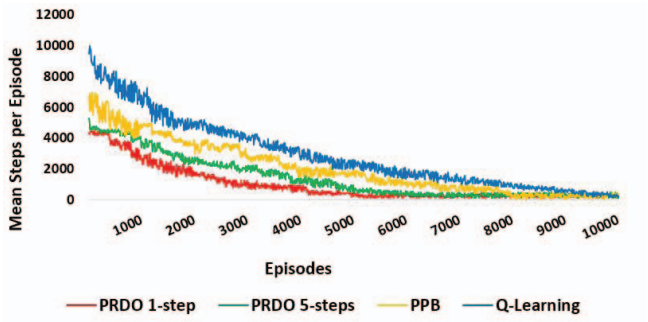

Fig. 5: The mean steps to goal for 10000 episodes during the learning process over 20 executions in the target tasks of TaxiWorld Domain. 2 passengers in the source tasks, 4 passengers in target task.

in the initial learning episodes, while also providing alternative decisions to the agent.

Usually, in the real world, the tasks are different and are more complex, then the idea of using too long n-step options may provide negative transfer instead helping the agent. Thus, we think that the parameter n should be balanced and changed according to the problem, maybe evaluating some way of similarity between tasks before.

VI. Conclusion and Next Steps

The main contribution of this work is the probabilistic reuse of combined object-oriented options to provide the agents good alternatives based on previous knowledge.

Our experiments in the *GridWorld* Domain and *TaxiWorld* Domain show that our framework is promising both for accelerating learning and guiding the agent with good solutions in RL tasks.

In the future, a good way to extend this approach is to better evaluate the initial solutions abstracted by the *Portable Options Discovery* approach. A possible way to do that is using the same idea of equation 3 from our approach instead of using the average of the sum of Q-Values to create the abstract state. Also a parameter or metric to define the number of steps n automatically when reusing the options would be interesting.

Another possibility is to evaluate the approach in more complex tasks, such as continuous [12], [25], [26] tasks where [11] has been shown to perform poorly; and multiobjective ones [14], [27], where they have not been evaluated yet. Many other option-discovery methods could also be tested and compared, and we believe they would fit very well in our framework. Finally, how to adapt our method to enable the transfer of learned options across different domains is still an open problem.

Acknowledgment

We are grateful for the support from the CEST Group, CNPq, grants 311608/2014-0, 307027/2017-1 and 425860/2016-7, and São Paulo Research Foundation (FAPESP), grants 2015/16310-4, 2016/21047-3, and 2018/00344-5.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
[2] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural computation*, pp. 215–219, 1994.
[3] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Springer, 2006, pp. 363–372.
[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[5] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *The Journal of Machine Learning Research*, pp. 1633–1685, 2009.
[6] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, pp. 181–211, 1999.
[7] M. V. Butz, S. Swarup, and D. E. Goldberg, "Effective online detection of task-independent landmarks," *Urbana*, vol. 51, p. 61801, 2004.
[8] A. McGovern, R. S. Sutton, and A. H. Fagg, "Roles of macro-actions in accelerating reinforcement learning," in *Grace Hopper celebration of women in computing*, vol. 1317, 1997.
[9] M. Pickett and A. G. Barto, "Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning," in *ICML*, 2002, pp. 506–513.
[10] M. L. Koga, V. F. Silva, and A. H. R. Costa, "Stochastic abstract policies: Generalizing knowledge to improve reinforcement learning," *Cybernetics, IEEE Transactions on*, pp. 77–88, 2015.
[11] D. S. Bernstein, "Reusing old policies to accelerate learning on new mdps," Citeseer, Tech. Rep., 1999.
[12] K. Subramanian, C. Isbell, and A. Thomaz, "Learning options through human interaction," in *IJCAI*. Citeseer, 2011.
[13] N. Topin, N. Haltmeyer, S. Squire, J. Winder, M. desJardins, and J. MacGlashan, "Portable option discovery for automated learning transfer in object-oriented markov decision processes." in *IJCAI*, 2015, pp. 3856–3864.
[14] R. C. Bonini, F. L. Silva, E. Spina, and A. H. R. Costa, "Using options to accelerate learning of new tasks according to human preferences," in *AAAI Workshop Human-Machine Collaborative Learning*, 2017, pp. (1–8).
[15] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture." in *AAAI*, 2017, pp. 1726–1734.
[16] F. Fernandez and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *AAMAS*, 2006, pp. 720–727.
[17] R. Glatt, F. L. Silva, and A. H. R. Costa, "Case-based policy inference for transfer in reinforcement learning," in *Workshop on Scaling-Up Reinforcement Learning at ECML*, 2017, pp. 1–8.
[18] F. L. Silva, M. E. Taylor, and A. H. R. Costa, "Autonomously Reusing Knowledge in Multiagent Reinforcement Learning," in *IJCAI*, 2018.
[19] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.
[20] C. Diuk, A. Cohen, and M. L. Littman, "An Object-oriented Representation for Efficient Reinforcement Learning," in *ICML*, 2008, pp. 240–247.
[21] F. L. Silva and A. H. R. Costa, "Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description," in *AAMAS Workshop on Transfer in Reinforcement Learning (TiRL)*, 2017.
[22] R. Glatt, F. L. Silva, and A. H. R. Costa, "Towards knowledge transfer in deep reinforcement learning," in *BRACIS*. IEEE, 2016, pp. 91–96.
[23] J. Macglashan, "Multi-source option-based policy transfer," Ph.D. dissertation, Catonsville, MD, USA, 2013.
[24] M. G. Madden and T. Howley, "Transfer of experience between reinforcement learning environments with progressive difficulty," *Artificial Intelligence Review*, vol. 21, no. 3-4, pp. 375–398, 2004.
[25] G. Konidaris and A. G. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," in *NIPS*, 2009, pp. 1015–1023.
[26] E. Brunskill and L. Li, "Pac-inspired option discovery in lifelong reinforcement learning," in *ICML*, 2014, pp. 316–324.
[27] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley, "A survey of multi-objective sequential decision-making," *CoRR*, 2014.