Learning Relational Options for Inductive Transfer in Relational Reinforcement Learning

Tom Croonenborghs, Kurt Driessens, and Maurice Bruynooghe

K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, B-3001 Leuven,
{ Tom.Croonenborghs, Kurt.Driessens,
Maurice.Bruynooghe}@cs.kuleuven.be

Abstract. In reinforcement learning problems, a learning agent has the task of learning a good or optimal strategy from interaction with his environment. At the start of the learning task, the agent usually has very little information. Therefore, when faced with complex problems that have a large state space, learning a good strategy might be infeasible or too slow to work in practice. One way to overcome this problem, is the use of guidance to supply the agent with traces of "reasonable policies". However, in a lot of cases it will be hard for the user to supply such a policy. In this paper, we will investigate the use of transfer learning for Relational Reinforcement Learning problems. The goal of transfer learning is to accelerate learning on a target task after training on a different, but related, source task. More specifically, we introduce an extension of the options framework to the relational setting and show how one can learn skills that can be transferred across similar, but different domains. We present some preliminary experiments showing the possible advantages of using relational options for transfer learning.

Keywords: Relational Reinforcement Learning, Transfer Learning, Options

1 Introduction

In reinforcement learning [12], an agent can observe its world and perform actions in it. The agent's learning task is to maximize the reward he obtains. In order to enable agents to learn in larger and more complex problem domains, abstraction has been an important factor. One important class of abstractions consists of hierarchical methods which focus on abstraction over the sequential and temporal aspects of a task [1]. Another direction of abstraction is relational reinforcement learning [14] which focuses on using relational representations for both the world (i.e., states and actions) and the learned policies.

One of the difficulties that remain is that at the start of the learning task, the agent has no or little information and is forced to do random exploration. As a consequence, learning can become infeasible or too slow in practice for complex domains.

One of the approaches to tackle this problem is the integration of guidance in reinforcement learning [5]. In this approach, traces of "reasonable policies" are used to overcome the problem of forced random exploration. The drawback of this approach is that the user still needs to provide such a "reasonable policy" that will provide the learning agent with some positive reinforcement.

Another approach that targets this problem and which has received a lot of attention recently is inductive transfer or transfer learning. Transfer learning is concerned with learning in one task to benefit learning in different but related tasks. More specifically, in a reinforcement learning context, the added effects of transfer learning can help the learning agent to learn a new (but related) task faster, i.e., with a smaller amount of training experience.

In this paper, we want to investigate the feasibility and benefit of using *hierarchical relational abstractions* for inductive transfer in reinforcement learning. The options framework is used for hierarchical abstractions [13]. Options are macro-actions that execute until some termination condition is satisfied and have been shown to be well suited to build high-level skills [9]. We introduce relational options as a combination of options with relational abstractions and use them to model skills that can be transferred across similar, but different domains.

Our contribution is threefold. First, we extend the options framework to the relational setting and show the benefits of such relational options. Second, we propose a method to learn options that can be used to transfer knowledge across different reinforcement learning domains using the framework of relational options. Third, we present some preliminary experiments to evaluate if skills learned as options in previous tasks can help the reinforcement learning agent in more difficult tasks.

2 Background

2.1 Relational Reinforcement Learning and the Blocks World

Reinforcement Learning (RL) [12] is often formulated in the formalism of *Markov Decision Processes* (MDPs).

Definition 1. *An MDP can be characterized by a state space S, an action space A, a state transition function T and an immediate reward function R. The transition function $T : S \times A \times S \rightarrow [0, 1]$ defines a probability distribution over the possible next states: $T(s, a, s')$ denotes the probability of landing in state s' when executing action a in state s. The reward function $R : S \times A \rightarrow \mathbb{R}$ defines the reward for executing a certain action in a certain state.*

The task of reinforcement learning consists of finding an *optimal policy* for a certain MDP, which is (initially) unknown to the RL-agent. As usual, we define it as a function of the discounted, cumulative reward, i.e. find a policy $\pi : S \rightarrow A$ that maximizes the value function: $V^{\pi}(s) = E_{\pi}[\sum_{t=0}^{\infty} \gamma^i R(s_t, \pi(s_t))|s_0 = s, s_{t+1} = \pi(s_t)]$, where $0 \leq \gamma < 1$ is the *discount factor*, which indicates the relative importance of future rewards with respect to immediate rewards.

The RRL-system [6] applies Q-Learning in relational domains, by using a relational regression algorithm to learn a Q-function that approximates the quality of executing a certain action in a certain state. The quality value for executing action a in state s with reward $R(s, a)$ and resulting state s' is defined as $Q(s, a) \equiv R(s, a) + \gamma\, max_{a'} Q(s', a')$. Knowing these Q-values, an optimal policy π^* can be constructed as $\pi^*(s) = \operatorname{argmax}_a Q(s, a)$.

Throughout the paper, we will use the blocks world as an example application. We use a blocks world with a varying number of blocks, where blocks can only be stacked neatly on top of each other and the table or floor is of infinite size, i.e., it is always clear and ready to store an extra block. Consider a set of blocks $\{b_1, b_2, \ldots b_n\}$, every block b_i stands either on the floor (denoted $on(b_i, floor)$) or on some other block b_j (denoted $on(b_i, b_j)$) and on every block b_i there is either exactly one other block or it is clear (denoted $clear(b_i)$). The agent can take a clear block b_i and put it on another clear block b_j or on the floor (denoted $move(b_i, b_j)$ or $move(b_i, floor)$ respectively).

2.2 The options framework

The theory of options has been introduced by Sutton et al. [13]. An option can be viewed as a subroutine, consisting of an option policy that specifies which action to execute for a subset of the environment states, an initiation set consisting of all states in which the option can be initiated and a termination condition, specifying when the option terminates. Note that an option is not just a sequence of actions, but a closed-loop policy taking actions depending on changes in the world.

More formally, an option o consists of three parts:

$$\begin{aligned} \mathcal{I}_o \subseteq S : S &\mapsto \{0,1\} \\ \beta_o : S &\mapsto [0,1] \\ \pi_o : \mathcal{I}_o \times A &\mapsto [0,1] \end{aligned}$$

with $\mathcal{I}_o$ the initiate set which specifies the states in which the option can be initiated, β_o the termination condition which specifies the probability of terminating in state s for all $s \in S$ and π_o the *option policy* , which specifies the probability of executing a in state s for all $a \in A, s \in \mathcal{I}_o$. Since an option policy can also invoke other options, creating hierarchical structures, the action space A is extended to be the set of all options and primitive actions.

To update the Q-value of an option o that is started in state s and terminated in state s', the following equation can be used (similar to the one for primitive actions): $Q(s,o) \equiv R(s,o) + \gamma^d max_{o'} Q(s', o')$ with d the duration of the option (i.e. the number of time steps between s and s').

2.3 Transfer Learning

One of the motivations for relational reinforcement learning is that by using parameterized goal and policy descriptions it allows learned results to be applied in similar, but different worlds. This transfer of knowledge is however limited to learning problems that exhibit the same structure, only differing in the identity of certain objects or the number of objects involved [4].

Recently, several approaches have been proposed to transfer knowledge between different propositional reinforcement learning tasks. Often, a user-defined mapping is used to relate the new task to the task for which a policy was already learned.

Some approaches use the learned knowledge to aid exploration in new (and usually more difficult) tasks. Examples of these are [10] and [8]. The most important drawback of these methods is that a lot of useful information is lost when only using the transferred knowledge for exploration.

Perkins and Precup [11] investigate the use of non-relational options to transfer knowledge. They however only study the scenario where the agent has to solve different tasks drawn from a given distribution in which the agent does not know which task he has to solve during a certain episode. The approach of [9] also uses options to transfer knowledge where the need for a mapping between different tasks is avoided by introducing a so-called "agent-space". This space is generated by a feature set that is present and retains the same semantics across successive problem instances.

The first approach that uses the generalizing power of relational learners is [15] where a relational rule learner is used to generate advice to speed up reinforcement learning. This advice is incorporated into the new task by adding the information about Q-values as soft-constraints to the linear optimization problem that approximates the Q-function for the next task which means that the advice needs to be propositionalized again for the new task. All advice is added to one big optimization problem, while the hierarchical abstractions in our approach are a first step towards a more modular representation of the policy.

3 Relational Options

The options framework can easily be extended to the relational setting by using a relational state and action space. We represent a relational policy by a set of rules of the following form: $\mathcal{C_S} : \mathcal{C_{SA}} \rightarrow \mathcal{A}$, with $\mathcal{C_S}$ a conjunction of tests that only involve the state space, $\mathcal{C_{SA}}$ a conjunction of tests on the state-action space and $\mathcal{A}$ the action predicted by this rule where the arguments are set by $\mathcal{C_{SA}}$. When a policy needs to determine which action to execute in a state, it searches for the top-most rule for which $\mathcal{C_S}$ is satisfied and an action $\mathcal{A}$ exists such that $\mathcal{C_{SA}}$ holds. Action $\mathcal{A}$ (with its arguments set according to $\mathcal{C_{SA}}$) will be predicted by the policy. Note that a distinction between $\mathcal{C_S}$ and $\mathcal{C_{SA}}$ is not strictly necessary. The predicate $s/1$ binds its argument with the current state and the $goal\_$-predicates are used to query the goal information: the $goal\_on/2$ predicate succeeds if the current goal of the agent is $on(A, B)$ and it will bind A and B with its arguments. Consider as an example an optimal policy for the $on(A, B)$ goal:

$$\begin{cases} s(S), goal\_on(A,B), clear(S,A), clear(S,B) : true \rightarrow move(A,B) \\ s(S), goal\_on(A,B), clear(S,A) : above(S,X,B), clear(S,X) \rightarrow move(X, floor) \\ s(S), goal\_on(A,B), clear(S,B) : above(S,X,A), clear(S,X) \rightarrow move(X, floor) \\ s(S), goal\_on(A,B) : above(S,X,A), clear(S,X) \rightarrow move(X, floor) \end{cases}$$

Although this relational extension of options is straightforward, it offers some advantages over regular options. Not only can they deal with relational worlds, it is also easy to extend them to a parameterized setting by expressing the policy in terms of variables instead of referencing concrete objects and by making it rely on the structural aspects of the task. Another advantage is that by having parameterized options, it also becomes possible to have recursive calls within the option policy.

Consider as an example an option that clears a certain block, instead of defining or learning a different option for every block that occurs in the world, it is possible to define one generalized option that can clear any block by introducing a variable as parameter of the option (as illustrated in Example 1). Note that this option does not specify a sequence of actions but a closed-loop policy deciding every time step which action to execute as long as the option is not terminated.

Example 1 (option for $clear(X)$).

$$\mathcal{I}_{clear(X)} : s(S) \mapsto 1$$

$$\beta_{clear(X)} : \begin{cases} s(S), clear(S,X) \mapsto 1 \\ s(S), \neg clear(S,X) \mapsto 0 \end{cases}$$

$$\pi_{clear(X)} : \begin{cases} s(S), goal\_clear(X) : on(S,Y,X), clear(S,Y) \rightarrow move(Y, floor) \\ s(S), goal\_clear(X) : on(S,Y,X) \rightarrow clear(Y) \end{cases}$$

4 Relational Skill Learning

One of the main difficulties in a lot of traditional transfer learning approaches is that in order to transfer knowledge one needs to provide a mapping from the source domain to the target domain. This problem is already partially overcome by using a relational representation since this allows us to abstract over specific object identities and even the number of objects involved.

In this paper we would like to transfer knowledge in the form of skills. Since we represent these skills as relational options, we need to specify the initiation set, the termination condition and the policy. To learn a specific option, the most straightforward approach would be to set the initiation set to $true$, the termination condition such that it is satisfied when a certain goal is reached and learn the policy with a standard relational reinforcement learning algorithm.

We take a slightly more advanced technique, motivated by the following observations: First of all, we would like to extract different skills from a single learning experience. Secondly, note that the Q-function contains more information than is actually needed for the optimal policy, i.e. it only requires a mapping from a state to the best action. Since the Q-function in a sense models the distance to reward, it does not always give the best generalization to new domains (with e.g. a different number of objects). Moreover, we would like the policy of the learned skill to be as interpretable as possible. Another disadvantage of traditional Q-learning approaches is that when an action needs to be predicted for a given state, an iteration is needed over all possible state-action pairs for that state to predict the action that results in the highest Q-value in case of a greedy policy. Since in complex domains with a lot of objects, the number of possible actions can be very large, we would like to avoid this iteration over all possible actions.

Therefore, we propose the following approach: The initiation set and termination condition can in the simplest case be set as specified above. Since the learned policy will not always be optimal it could be the case that the option never terminates. To avoid this scenario, the termination condition is changed so that every non-goal state has a non-zero probability of terminating the option. A skill policy for a certain task is built as follows where the first three steps are similar to the P-learning approach [6]:

1. use an RRL algorithm to learn to solve this task
2. create training examples of state-action pairs labeled as policy-based or not
3. use the TILDE system [2] to learn a relational decision tree that predicts whether or not the action will be executed by the policy
4. extract a relational policy

To create the dataset for TILDE, we create a number of random states and check which action would be executed by the learned policy. This action gets the label "policy action", other actions the label "non-policy action". For the moment we do not use any sampling techniques, but in the future we would like to investigate possible sampling techniques as well as different schema to create learning examples. One possibility is to only include examples of state-action pairs for which the Q-value is significantly better or worse than other state-action pairs for the same state (similar to the approach taken in [15]).

The learned decision tree for this binary classification problem predicts if, given a certain state and action, the action is the one that would be executed by the policy (or the probability if we use probability trees). Although in future work we plan to investigate extraction schemas that obtain stochastic policies from these probability trees, at the moment we only look at the majority class in each of the leaves. This means we can do some bottom-up post-pruning if both the 'yes' and 'no' branch predict the same majority class. Figure 1 shows an example of a policy tree for the $clear(X)$ skill.

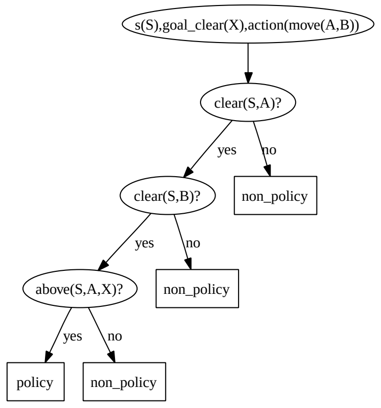

Fig. 1. Policy tree for $clear(X)$ that expresses that an action is a "policy action" if both arguments of the move action are clear and the agent moves a block above the one that needs to be cleared.

To avoid the action iteration, we extract from this tree only those rules that predict the 'policy action' class. These rules give the constraints on the action to be part of this policy. If we do this for the tree in Figure 1, we only obtain the following rule: $s(S), goal\_clear(X), action(move(A, B)), clear(S, A), clear(S, B), above(S, A, X) \rightarrow policy$. A policy is obtained by transforming each rule to the form $\mathcal{C}_{\mathcal{S}} : \mathcal{C}_{\mathcal{S}\mathcal{A}} \rightarrow \mathcal{A}$, the order of the rules is not important since the decision tree partitions the state-action space. For computational reasons it could be interesting to first consider rules with the least negations in them. A default rule is also added that predicts a random action in case none of the other rules apply. For the policy tree in Figure 1, this gives the following relational policy:

$$\begin{cases} s(S), goal\_clear(X) : clear(S,A), clear(S,B), above(S,A,X) \rightarrow move(A,B) \\ s(S), goal\_clear(X) : random\_block(X), random\_block(Y) \rightarrow move(A,B) \end{cases}$$

There are different approaches to extract more than one skill. One possibility is to repeat the above procedure with different language bias settings, e.g. to obtain a different level of generalization. For instance the approach of [9] can be modeled with our approach by using appropriate language bias settings for both “problem-space” and “agent-space” skills. Another approach would be to specialize the initiation set of an option. For instance if the top test of the policy tree is not selective on the arguments of the action, the policy will be different for the two partitions of the state space.

The learned skills do not necessarily have to be subgoals in the new task as is the case in our blocks world example. We expect that transferring knowledge in the form of options is also useful in other cases. Consider as an example learning a task in a stochastic environment where the agent already learned a skill to solve the same task in a non-stochastic version of this environment. This skill will then probably be most interesting in the beginning of the learning task as a means of good exploration, while the learning agent might find better policies when he collects more information.

5 Preliminary Experiments

We consider a target task in the blocks world with the $on(A, B)$-goal, i.e. the agent receives a reward iff block A is directly on top of block B. The number of blocks is varied between 5 and 15 every episode. During exploration, the learning agent is allowed 10 steps more than needed to reach the goal. To evaluate the agents’ learning behavior, 100 tests are performed following a greedy policy checking the percentage of episodes in which an optimal path is followed. The RRL-TG[3] method is used in all experiments to approximate the Q-function. Figure 2 shows the results averaged over 5 different runs.

First of all, we will compare the learning behavior of a standard RRL agent using only primitive actions and an agent that can also use the $clear(X)$ option as defined in Example 1. As expected, Figure 2 clearly shows an improvement in learning behavior for the agent that can use this skill.

Next, we would like to investigate how difficult it is to learn such a $clear(X)$ skill. To learn this skill, we set up a blocks world environment with the $clear(X)$-goal in which the number of blocks is varied between 5 and 10. The Q-function learned by the agent after 100 episodes is taken to create training examples. This dataset is created using states from 50 new episodes[1]. The policy trees learned in the 5 different runs all

[1] Instead of using episodes, it is also possible to create the dataset using random states, i.e. episodes of length one

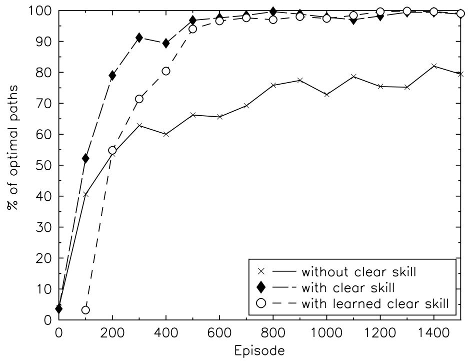

Fig. 2. Results in the blocks world with $on(A, B)$ goal and the $clear(X)$ skill

looked similar to the one shown in Figure 1. They only differed in the order of the tests and the distributions in the leaves. Note that by only looking at the majority class noise in the Q-function can be filtered out. As a result the learned policy tree is optimal, while the policy based on the Q-function was not. Of course, by only looking at the majority class, it could also happen that important information is filtered out. We will address these issues in future work.

Since in general we do not know whether the learned skill is optimal, we set the termination condition so that it terminates for states in which $clear(X)$ holds and with a probability of 0.05 otherwise. Figure 2 shows the learning behavior of a learning agent using primitive actions and this learned option. To compensate for the 100 training episodes in the $clear(X)$ task we shifted this learning curve to make a fair comparison. Since an optimal policy is learned for this $clear(X)$ skill, the learning behavior is similar to the one with the user-defined $clear(X)$ skill and significantly better than the learning behavior of the agent with only primitive actions.

6 Conclusions and Further Work

In this paper we presented an extension of the options framework to the relational setting. We have also shown how this framework can be used to learn relational skills that can be transferred across similar, but different tasks. Some very preliminary experiments motivate the use of such learned skills in RRL problems.

In future work, we will perform a more in depth analysis of the approach we presented. Currently, we have not considered the problem of determining which skills to learn. Since we learn parameterized options this is less of a problem, since it will often be possible to learn a skill for every predicate in the state representation (e.g. usually $clear/1$ and $on/2$ in the blocks world).

When learning these skills is hard, one could consider the approach of Fern et al. [7] to generate extra learning experience by using relational abstractions to create artificial goals in non-goal states. E.g. in every state there will be a clear block, so it is possible to assign the clearance of that block as a goal for the agent to create training experience for the $clear(X)$ goal.

Furthermore, at the moment we only consider the use of skills learned in previous tasks. A natural extension would be to focus on a hierarchical decomposition and learn new skills in the current task and to modify previous learned skills while learning in the new task.

Acknowledgments

Tom Croonenborghs is supported by the Flemish Institute for the Promotion of Science and Technological Research in Industry (IWT).

References

1. Andrew Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete-Event Systems journal*, 13:41–77, 2003.
2. H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
3. K. Driessens, J. Ramon, and H. Blockeel. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In L. De Raedt and P. Flach, editors, *Proceedings of the 12th European Conference on Machine Learning*, volume 2167 of *Lecture Notes in Artificial Intelligence*, pages 97–108. Springer-Verlag, 2001.
4. K. Driessens, J. Ramon, and T. Croonenborghs. Transfer learning for reinforcement learning through goal and policy parameterization. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning (Online Proceedings)*, 2006.
5. Kurt Driessens and Saso Dzeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, December 2004.
6. S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.
7. Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational Markov decision processes. *Journal of Artificial Intelligence Research*, 25:85–118, 2006.
8. Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727, NY, USA, 2006. ACM Press.
9. George Konidaris and Andrew Barto. Building Portable Options: Skill Transfer in Reinforcement Learning. In Manuela Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2895–900, Hyderabad, India, January, 6-12 2007.
10. Michael G. Madden and Tom Howley. Transfer of Experience Between Reinforcement Learning Environments with Progressive Difficulty. *AI. Rev.*, 21(3-4):375–398, 2004.
11. T. J. Perkins and D. Precup. Using Options for Knowledge Transfer in Reinforcement Learning. Technical Report UM-CS-1999-034, University of Massachusetts, MA, USA, 1999.
12. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.
13. R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
14. P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning*, 2004.
15. L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Skill acquisition via transfer learning and advice taking. In *Proceedings of the 17th European Conference on Machine Learning*, pages 425–436, 2006.