Transfer Learning for Reinforcement Learning through Goal- and Policy Parametrization

Kurt Driessens KURT.DRIESSENS@CS.KULEUVEN.BE
Jan Ramon JAN.RAMON@CS.KULEUVEN.BE
Tom Croonenborghs TOM.CROONENBORGHS@CS.KULEUVEN.BE
Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium

Abstract

Relational reinforcement learning has allowed results from reinforcement learning tasks to be re-used in other, closely related, tasks. This transfer of knowledge is made possible by the use of parameters in the representations of the task-description and the learned policy.

In this paper, we will give a description of the current state of the art of transfer learning with relational reinforcement learning, make some observations about the usefulness and limitations of this current state and discuss some directions for future research. We also present a first small step along one of those directions.

1. Introduction

Relational reinforcement learning (Džeroski et al., 1998; Džeroski et al., 2001) has received a lot of attention over the last few years. The use of relational representations for both the world (i.e., states and actions) and the learned value-functions and policies allows reinforcement learners to look at larger and more complex problem domains. It raised the level of possible applications of reinforcement learning substantially and, most importantly, allowed learned results to be applied in worlds or tasks closely related to, but different from, the world that was initially learned in. This was a first step of solving one of the biggest drawbacks of reinforcement learning, i.e., that a standard reinforcement learner has to be retrained from scratch if anything changes in the specification (task or representation related) of the learning problem. Because a reinforcement learner constructs a policy through interaction with its environment, this is usually an expensive operation.

The advantages of relational reinforcement learning are mainly due to the use of goal- (and policy-) parametrization in the contruction of the policy[1]. By forcing the learned policy to be expressed in terms of variables instead of allowing it to reference concrete objects and by making it rely on the structural aspects of the task, it becomes natural and easy to transfer the learned knowledge to learning problems that exhibit the same structure, but might differ in respect of the identity of certain objects or even the number of objects involved.

In this paper, we give some illustrations of how parametrization can be handled in relational reinforcement learning and discuss the opportunities it creates for transfer of knowledge to related tasks as well as the limitations it still has to deal with. We also propose some directions that can be investigated to overcome these limitations and a possible first step along one of those directions.

The rest of the paper is structured as follows: section 2 discusses the current possibilities for transfer learning with relational reinforcement learning. Section 3 makes some observations on possible directions for future work. Section 4 concludes.

2. Current State of the Art

To be able to illustrate the state of the art in goal- and policy-parametrization, we will rely on the blocks world as an application example. The blocks world has been adopted as a standard testbed for relational

[1] Another way of looking at the abstractions obtained with relational representations is to start from a logical abstraction of the MDP, also referred to as RMDP (for Relational MDP). We will not discuss that viewpoint in this paper. Interested readers can consult (van Otterlo, 2005) for a very recent overview of work in relational reinforcement learning starting from that approach.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

reinforcement learning. 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.

Instead of solving problems such as: "Put the block with id 3 on the block with id 5", relational reinforcement learning is used to learn goals such as: "Put block X on block Y", where X and Y are variables. This parametrization of goals automatically allows the learned results to be applied to the problem of stacking any two blocks in the studied world. Other "relational" goals include for example "Build one tall stack" where the order of the blocks in the stack does not matter. Referencing specific blocks in a policy for this goal makes little sense.

To learn these parametrized goals, relational reinforcement learning uses representations for the utility-functions or policies that do not refer to specific objects. Instead it uses the structure of the problem to predict utility values or to decide which actions to take. Two different approaches have been used to handle this.

One direct way of referencing blocks without using their specific identity, is to build a logical model of the value-function or policy and restricting the generalization algorithm to refer to objects only indirectly, through the use of variables. One example of such an approach is the RRL-TG algorithm (Driessens et al., 2001), which uses a first-order decision tree to approximate the value function or the policy.[2] Figure 1 shows an example of a policy in the shape of a first-order decision tree for the "on(A,B)" goal that works for any number of blocks. As well as solving the "on"-problem for any two blocks in the world, the abstraction made by such a parametrized policy, also works in worlds with a different number of objects. By focussing only on the blocks that are featured in the goal or the current action the constructed policy becomes indifferent to other objects (or distractions) that might or might not be present. However, to be able to do this, the use of the *above(X,Y)* predicate was required. While RRL-TG will be able to construct an optimal policy using only *clear(X)* and *on(X,Y)* predicates, the resulting tree will not transfer so easily to worlds with different numbers of blocks. The success of the knowledge transfer will often depend on the representations used.

Other forms of abstraction used for relational reinforcement learning make use of structural representations of states, actions and goals, that explicitly try to remove object identity. Examples of this are instance based methods that use relational distances (Driessens & Ramon, 2003) or kernel approaches that work of graph-representations (Gärtner et al., 2003). The relational distance, for example, can take possible mappings for related blocks in different states (or even worlds) into account. This requires the design of suitable distances or kernels, but this is not much different from the definition of background predicates such as *above* in the previously discussed example.

[2]Other approaches that use e.g. first-order rules exist as well. Examples include the work by Fern et al. (2003) and Kersting et al. (2004).

3. Observations and Directions for Further Research

The previous section showed how transfer of knowledge is incorporated in relational reinforcement learing. Given a good goal-parametrization and the correct representation language, this becomes almost trivial and the generalized policy can be learned at almost no extra cost. These parametrized policies seem well suited for use in e.g. hierarchical reinforcement learning, where they can be used as parametrized options (Sutton et al., 1999).

However, the discussed transfer is limited to problems that are very similar, i.e. the goals need to be structurally identical. As soon as the structure of the goal changes (according to the used representational language), results of previous learning tasks are a lot more difficult to transfer to the new task.

It is possible to apply the same approach as discussed before to further abstract reinforcement learning problems. Where parametrization is up to now only used to abstract over specific object identities, it could also be used to abstract over structural differences in tasks. Instead of starting the tree of Figure 1 with the predicate *goal_on(A,B)*, we could have initiated the root with the predicate *goal(X)*. This would make abstraction of the exact structure of the goal, so that e.g. the reinforcement learning algorithm could learn both "Build one large stack" and "Put block X on block Y" in the same tree.

A trade-off emerges here which allows for some transfer of difficulty between two steps of transfer learning: the initial learning process and the transfer of knowledge to a related task. Learning a policy for a very general problem could be hard, but would greatly reduce the work that still needs to be done when transferring the learned knowledge to another task. On the other hand, learning a more specific policy will reduce the complexity of the initial step, but will require more

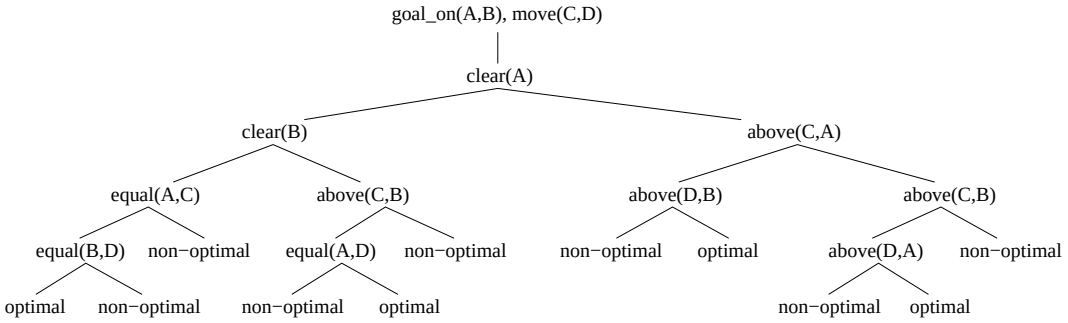

Figure 1. A first-order decision tree describing a general policy for stacking any two specific blocks.

work during the transfer. We will briefly look at the specifics of both.

3.1. Parametrization of the Goal Structure

Learning a general policy for a set of tasks in related worlds seems like the holy grail of learning in control problems. Depending on the generality of the set of tasks, this problem ranges from relatively easy to very hard.

Although some existing approaches already try to solve this kind of problems — for example in the blocks world, by using predicates such as *inGoal(X)* and *inPlace(X)* (Khardon, 1999) — it is our belief that a taxonomy will have to be defined on the encountered tasks. Such a taxonomy will allow the generalized policy to discover e.g. actions that are usefull in a large subset of tasks, which tasks are subtasks of others, etc. For the algorithm to be able to exploit such a taxonomy to the fullest, it will beneficial to relate the representation of the goal, to the representation of the states and actions. A goal description using *onTopOf(X,Y)* predicates will be of less use to a learning agent using the *on(X,Y)* predicate in state descriptions if it doesn't know that the two are related (identical in this case).

3.2. Partial Reuse of Learned Results

Instead of trying to solve all possible learning problems in the first step, one can also take a look at related work in the field of *Theory Revision*. The biggest difference in the problem definition of transfer learning and theory revision is that for transfer learning, it is often known that the results from the first step will have to be revised later on in the experiment. This knowledge can be used to develop algorithms that create more easily transformable models.

Since learning policies for a large variety of tasks comes close to learning actual programs, transfer learning could benefit from modular representations of policies, where some sub-modules could be usefull for several tasks. This relates again to hierarchical reinforcement learning and the use of options to enable the transfer of learned skills to related tasks.

A Small First Step To be able to recycle some of the obtained knowledge when the structure of the problem (or the solution) changes, we need an algorithm that can re-use parts of the learned structure and alter those parts that no longer apply. This is difficult to imagine for instance-based approaches such as the discussed RRL-RIB and RRL-KBR systems, that use implicit structural knowledge that can not be directly altered. However, the explicit and interpretable structural knowledge contained in a first-order regression or classification tree such as built by the RRL-TG algorithm, can be addressed and altered where necessary.

For this, tree-restructuring operators are necessary that can be applied to first-order decision trees. Because of the dependencies between tests in different nodes of a first-order tree, this is not trivial. Four tree-restructuring operators that can be defined for first-order trees are the following:

1. **Splitting a leaf**: This operator splits a leaf into two subleafs, using the best suited test. This is the only operator used by standard (non-restructuring) TDIDT algorithms such as the TG algorithm mentioned above.
2. **Pruning a leaf**: This is the inverse operator of the first. When predictions made in two sibling-leafs (leafs connected to the same node) become similar, this operator will join the two leafs and remove the internal node.
3. **Revising an internal node**: The revision operator is a bit more complex than the two previous ones. When a chosen test in a node becomes non-optimal and it turns out that another test would

be better suited to distinguish between examples at that level, the dependencies between tests in first-order trees make it impossible to make a straightforward swap of the two tests. Dependencies between tests in first-order trees originate from the use of variables. Any node can introduce new variables that can be referenced elsewhere in the tree. Removing such a node can change the semantics of the tests referring to the (also) deleted variables. E.g. the semantics of the tree shown in Figure 1 would change substantially if we removed the initialization conjunction at the top. One option is to prune the tree completely starting from the deleted test, putting a new two-leaf tree with the new test at the root in its place, but a lot of information would be lost in this way. Instead, we will opt to use a copy of the subtree that starts with the old test as both the left and the right subtree of the new node. This will allow the re-use of the information stored in the previously built subtree, and, if the new test is chosen wisely, a large portion of the two subtrees will be pruned again at later stages by operators 2 and 4.

4. **Pruning a subtree**: This operator is related to the previous one, but will shrink the tree instead of enlarging it. This operator will be called when a node can (or should) be deleted. Because of the dependencies between tests in a first-order tree, the choice of subtrees to replace the original tree starting at the deleted node is limited to the right-side one. As before, the left subtree can contain references to variable introduced by the test used in the deleted node.

To use these restructuring operators, an algorithm will have to be designed that collects suited statistical evidence to decide when and where the operators should be applied.

4. Conclusions

We've presented the current state of the art in goal- and policy-parametrization in relational reinforcement learning. For structurally identical tasks (with structural identity defined in relation to the used representation) goal-parametrization offers a very easy way to do knowledge transfer in reinforcement learning.

For tasks with structural differences, we looked at expanding the level of goal-abstraction to incorporate different structures and at re-using partial knowledge by using approaches related to theory revision. We presented some initial thoughts on a first-order tree-restructuring algorithm that could be used for transfer learning in reinforcement learning tasks when only parts of the learned structure are appropriate for the new task.

References

Driessens, K., & Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 123–130). AAAI Press.

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. *Proceedings of the 13th European Conference on Machine Learning* (pp. 97–108). Springer-Verlag.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Džeroski, S., De Raedt, L., & Blockeel, H. (1998). Relational reinforcement learning. *Proceedings of the 15th International Conference on Machine Learning* (pp. 136–143). Morgan Kaufmann.

Fern, A., Yoon, S., & Givan, R. (2003). Approximate policy iteration with a policy language bias. *Proceedings of the Seventeenth Annual Conference on Neural Information Processing Systems*. The MIT Press.

Gärtner, T., Driessens, K., & Ramon, J. (2003). Graph kernels and Gaussian processes for relational reinforcement learning. *Inductive Logic Programming, 13th International Conference, ILP 2003, Proceedings* (pp. 146–163). Springer.

Kersting, K., van Otterlo, M., & De Raedt, L. (2004). Bellman goes relational. *Proceedings of the 21st International Conference on Machine Learning* (pp. 465–472).

Khardon, R. (1999). Learning to take actions. *Machine Learning*, *35*, 57–90.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDP's and semi-MDP's: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

van Otterlo, M. (2005). *A survey of reinforcement learning in relational domains* (Technical Report TR-CTIT-05-31). CTIT Technical Report Series, ISSN 1381-3625.