Reinforcement Learning for Relational MDPs

Martijn van Otterlo
TKI - Department of Computer Science
University of Twente - The Netherlands
otterlo@cs.utwente.nl

Abstract

In this paper we present a new method for reinforcement learning in relational domains. A logical language is employed to abstract over states and actions, thereby decreasing the size of the state-action space significantly. A probabilistic transition model of the abstracted Markov-Decision-Process is estimated to speed-up learning. We present theoretical and experimental analysis of our new representation. Some insights concerning the problems and opportunities of logical representations for reinforcement learning are obtained in the context of a growing interest in the use of abstraction in reinforcement learning contexts.

1 Introduction

There is general agreement nowadays that intelligent agents should be *adaptive*, i.e. capable of learning. Reinforcement learning (RL) (Sutton and Barto, 1998) is a very powerful paradigm for behavior learning, in which a learning agent performs actions to reach a particular goal state and gets rewards while doing that. The goal of the agent is to maximize its reward intake. For example, an agent can learn how to navigate through a maze quickly, by rewarding each step with a reward of -1, so that it will minimize the number of steps needed.

Starting to grow in the early 90's, RL is young as a research field. Algorithms for calculating value functions are understood relatively well these days. Much effort is being put on scaling up towards larger problems by discovering and using task *structure* e.g. by using hierarchy, temporal abstraction and *factored* representations. Another form of abstraction can be obtained by allowing more powerful representation *languages* (Otterlo, 2002).

Some issues in relational representations were recently addressed in both *decision theoretic planning* and (model-free) RL, but currently there is still large gap between RL - traditionally dominated by the use of propositional and attribute-value representations - and logical languages as traditionally used in the *agent* literature and the planning community (see also (Otterlo et al., 2003)).

In this paper we present a new method for RL using a relational representation that allows RL methods to be applied in environments that can be naturally represented in terms of *objects* and *relations*. We present a model of *relationally factored Markov decision processes* for these environments and we present a representational device called *CARCASS*. We also extend our approach by using learned world models to speed up learning. Using relational languages as representation in RL creates many opportunities but also generates interesting new problems, some of which we will also address.

This paper is structured as follows. Section 2 will introduce some basic notions after which in section 3 a model of RMDPs is discussed and our proposed abstraction, the *CARCASS*. This section also highlights an RL algorithm. Section 4 will then discuss a model-based extension and section 5 will present theoretical and experimental analysis of our method in two different domains. After that a selection of related work is discussed and the paper ends with conclusions and directions for further research.

2 Preliminaries

In the remainder of this paper we will use terminology of logical languages and Markov Decision Processes. In this section we will briefly introduce some main notions. For more details, see (Sutton and Barto, 1998) and (Bergadano and Gunetti, 1995). Furthermore, all examples concerning *blocks worlds* use the standard, intuitive environment where some blocks are on a *floor* (f), blocks can be moved around with $move(X,Y)$, denoting that block X is moved on top of Y, and where the goal is to reach a certain configuration of towers of blocks. Valid relations are $on(X,Y)$, i.e. block X is on Y and $cl(Z)$, i.e. block Z is *clear* and can be moved.

2.1 Logic

A *first-order alphabet* Σ is a set of predicate symbols p with arity $m \geq 0$, and a set of constants c. An *atom* $p(t_1, \ldots t_m)$ is a predicate symbol p followed by a bracketed m-tuple of terms t_i. A *term* is a variable X or a constant. A *conjunction* is a set of (possibly negated) atoms. The set of variables in a conjunction A is denoted $vars(A)$. A substitution θ for a conjunction A is a set assignments of terms to variables $\{X_1/t_1, \ldots X_n/t_n\}$ where X_i is in $vars(A)$ and all t_i are terms. A term, atom or conjunction is called *ground* if it contains no variables. *Background knowledge* (BK) is a set of *Horn Clauses* $H \leftarrow B$ where B is a conjunction (or $\emptyset$), H, the *head*, is an atom and $vars(H) \subseteq vars(B)$. Conjunctions (and Horn Clauses) are implicitly *existentially quantified.* A conjunction A entails a conjunction B, denoted $A \vdash_\theta B$, resulting in a substitution θ if there exists a proof of B from A using the BK as axioms. The *Herbrand base* of Σ, hb^Σ, is the set of all ground atoms which can be constructed with the predicate symbols and constants. A *first-order interpretation* is a subset of the Herbrand base.

2.2 Markov Decision Processes

A *Markov Decision Process* (MDP) is a tuple $M = \langle S, A, T, R \rangle$, where S is a set of states, A a set of actions, $T : S \times A \times S \to [0,1]$ a *transition function* and $R : S \times A \times S \to [0,1]$ a *reward function* . The set of actions *applicable* in a state $s \in S$ is denoted $A(s)$. A transition from state $i \in S$ to $j \in S$ caused by some action $a \in A(i)$ occurs with probability $T(i,a,j)$ and a reward $R(i,a,j)$ is received. T defines a proper probability distribution if for all states $i \in S$ and all actions $a \in A(i)$: $\sum_{j \in S} T(i,a,j) = 1$. A deterministic *policy* $\pi : S \to A$ for M specifies which action $a \in A$ will be executed when the agent is in some state $s \in S$, i.e. $\pi(s) = a$.

2.3 Value functions

Given some MDP $M = \langle S, A, T, R \rangle$, a policy π for M, and a *discount factor* $\gamma \in [0,1]$, a *state value function* $V^\pi : S \to \mathbb{R}$ represents the value of being in a state while following policy π, w.r.t. expected rewards. It can be shown that for each state $i \in S$: $V^\pi(i) = \sum_{a \in A(i)} \sum_{j \in S} T(i,a,j)[R(i,a,j) + \gamma \cdot V^\pi(j)]$. A similar *state-action value function* $Q^\pi : S \times A \to \mathbb{R}$ can be defined. A policy π^* is optimal if $V^{\pi^*}(s) \geq V^{\pi'}(s)\ \forall s \in S$ and $\forall \pi'$. Optimal value functions are denoted V^* and Q^*.

3 Relationally Factored MDPs

3.1 RMDP

Environments can be described in terms of *objects* and *relations* between objects. A first consequence is that the MDP is not described in terms of discrete states, or factored into propositions, but we need a *relationally factored* MDP (RMDP). In this section RMDPs will be defined, as well as a representational device useful for reinforcement learning. Let us first define RMDPs:

Definition 1 *A* Relationally factored *MDP (RMDP) M is a tuple $\langle P, A, D, T, R \rangle$ where D is a (finite) domain of objects, P is a (finite) set of predicate templates and A is a (finite) set of action templates. The state space S, $S(M)$, is defined as a unique*[1] *subset of the set of all first-order interpretations over P and D. The action set $A' = hb^{(A \cup D)}$. Given S and A', the transition function T and reward function R are defined as in section 2.2.*

For example, let $B3$ be the RMDP $\langle \{on/2, clear/1\}, \{move/2\}, \{a,b,c,f\}, T, R \rangle$ for a blocks world with three blocks (a,b,c) and a floor (f) One state in which all blocks are on top of each other is $\{on(a,b), on(b,c), on(c,f), clear(a)\}$. Remember that an interpretation is the set of ground relations that are true in some state. The action for moving block a from the top of the stack to the floor is $move(a,f)$. Rewards are 0 for every action unless resulting in a

[1]Usually an implicit background theory restricts S. For example, in the blocks world no block can be on top of itself, i.e. $\neg \exists s \in S. on(X,X) \in s$.

transition to a *goal* state, in which case the reward is 1.

The size of the state space for an RMDP can grow extremely large with only a modest number of relations. As an example, for blocks world with 5 blocks, $|S| = 501$ and for 10 blocks, $|S| \sim 59$ million. Considering the fact that even small RMDPs generate huge state spaces, some type of abstraction is needed to aggregate states and actions that are 'similar'.

Definition 2 *Let $M = \langle P, A, D, T, R\rangle$ be an RMDP and let BK be a set of Horn clauses. An* abstract state *$\tilde{s}$ is a conjunction over P, D and the heads in BK. An* abstract action *$\tilde{a}$ is an atom over A and D. The set $S(\tilde{s})$ of ground states* aggregated[2] *by an abstract state $\tilde{s}$ is $\{s \in S(M) \mid s \vdash_\theta \tilde{s}\}$.*

For example, in a blocks world containing three blocks a, b and c, the abstract state $\tilde{s} = on(A, B) \wedge (B, C) \wedge cl(A)$ aggregates over all states except the *unstack* state where all the blocks are on the floor.

A set of abstract states partitions the state space only if every state belongs to only one abstract state. It is possible to ensure that in a syntactic way, e.g. each ground state proves one unique abstract state. In this paper we choose a different solution. We can define an ordering on the abstract states, in a decision list fashion. By using such a simple ordering, a set of abstract states $\{\tilde{s}_1, \dots \tilde{s}_n\}$ induces a partition of $S(M)$ of equivalence classes of states $[\![\tilde{s}_1]\!], \dots, [\![\tilde{s}_n]\!]$ using $[\![\tilde{s}_1]\!] = S(\tilde{s}_1)$, and for $i = 2 \dots n$:
$[\![\tilde{s}_i]\!] = S(\tilde{s}_i) \setminus (\bigcup_{j=1}^{j=i-1} [\![\tilde{s}_j]\!])$
The goal of RL is learning an optimal policy. By using abstractions over states and actions, we can define Q-value functions for abstract states and actions, and define abstract policies for an RMDP. Action rules are the constituents of abstract policies. Let us first define an abstract action rule and explain what the procedural semantics are:

Definition 3 *An abstract action rule $\tilde{t}$ is a rule $\tilde{s} \rightarrow \tilde{a}$, (or $(\tilde{s}, \tilde{a})$) where $\tilde{s}$ is an abstract state, $\tilde{a}$ an abstract action and $var(\tilde{a}) \subseteq var(\tilde{s})$.*

[2]Throughout the paper we will assume that substitutions will map each variable to a unique domain object. In our implementation this is enforced by using extra atoms such as $not(A == B)$, with '==' in BK.

The semantics of an abstract action rule $\tilde{s} \rightarrow \tilde{a}$ is that if an agent is in some state $s \in [\![\tilde{s}]\!]$ then by choosing action $\tilde{a}$ it will perform some (nondeterministically chosen) action $a = \tilde{a}\theta$ where $s \vdash_\theta \tilde{s}$. In this paper we will assume that action a is chosen with probability $\frac{1}{n}$ with n being the total number of different substitutions θ. The set of state-action pairs aggregated by some abstract transition rule is defined by:
$[\![\tilde{s}, \tilde{a}]\!] = \{(s, a) \mid s \in [\![\tilde{s}]\!] \wedge s \vdash_\theta \tilde{s} \wedge a = \tilde{a}\theta\}$
The set of ground actions applicable in some state s relative to some abstract transition rule is defined by:
$\forall s \in [\![\tilde{s}]\!] : [\![\tilde{s}, \tilde{a}]\!]^s = \{a \mid (s, a) \in [\![\tilde{s}, \tilde{a}]\!]\}$ So, $[\![\tilde{s}, \tilde{a}]\!]^s$ is the set of actions that are 'similar' w.r.t. the abstraction level used. One cannot discern between the actions and an arbitrary action can be chosen from this set. A policy $\tilde{\pi}$ consists of a set of action rules, where the rules are ordered in the same fashion as in the above.

3.2 CARCASS Abstraction

In this section we will define a formalization of a *skeleton* of an RMDP, a *CARCASS* (**C**ompact **A**bstraction using **R**elational **C**onjunctions for **A**ggregation of **S**tate-action **S**paces). A CARCASS defines a relational abstraction in terms of pairs of an abstract state and a set of abstract actions applicable in that abstract state. A CARCASS restricts the state-action space for the learner considerably.

Definition 4 *Let $M = \langle P, D, A, T, R\rangle$ be an RMDP. A* CARCASS *$\tilde{C}$ for M, denoted $\tilde{C}(M)$ is a structure $\langle \widetilde{sa}, \leq, BK\rangle$ where $\widetilde{sa}$ is a set $\{(\tilde{s}, \tilde{A})\}$ where each $(\tilde{s}, \tilde{A})$ is an abstract state $\tilde{s}$ with a set $\tilde{A}$ of applicable, abstract actions, $\leq$ is an ordering on the abstract states and BK is a set of predicate definitions, i.e. a set of Horn clauses. For all $(\tilde{s}, \tilde{A}) \in \widetilde{sa}$ and $\tilde{a} \in \tilde{A}$; $var(\tilde{a}) \subseteq var(\tilde{s}_i)$. The set of abstract states in $\tilde{C}(M)$ is denoted $\tilde{S}(\tilde{C})$. All $\tilde{s} \in \tilde{S}(\tilde{C})$ are conjunctions over P and the heads in BK.*

In figure 1 we can see an example of a CARCASS for a three blocks world. The three states abstract over block names, e.g. the abstract states model exactly the three possible *configurations* consisting of three blocks. In each state a number of abstract actions are possible. Note that in the second state, we can easily replace the 6 abstract actions by only one, $move(A, B)$

```
(1) on(A,B),on(B,C),on(C,D),on(D,f),cl(A)
        move(A,f)
(2) on(A,B),on(B,C),on(C,f),on(D,f),cl(A),cl(D)
        move(A,f),move(A,D),move(D,A)
(3) on(A,B),on(B,f),on(C,D),on(D,f),cl(A),cl(C)
        move(A,f),move(A,D)
(4) on(A,B),on(B,f),on(C,f),cl(A),cl(C)
        move(A,f),move(A,C),move(C,A)
(5) on(A,f),on(B,f),cl(A),cl(B)
        move(A,B)
```

Figure 1: CARCASS $\tilde{C}_1$ for a 4-blocks world. (Linear ordering on states and $BK = \emptyset$)

because of substitutions. Further simplifications can be made by making use of the state ordering. By supplying background knowledge definitions, the state abstractions can be very powerful. For example, it is possible to define a predicate $towerHeight(A, H)$ relating the top of a tower, denoted A, to the tower height H.

CARCASS's can be used to learn an abstract Q-function $\tilde{Q}$ for abstract state-action pairs. We will first discuss Q-learning for our abstract representation and give a general algorithm.

Algorithm 1 Main RL Loop

```
Require: environment is initialized and
    all Q-values initialized to 0.
 1: for all episodes do
 2:   Initialize start state
 3:   while NOT ((end of episode) OR
      (maximum number of steps reached)) do
 4:     $s$ is current ground state
 5:     find: $\tilde{s} \in \tilde{S}(C)$ for which $s \in [\![\tilde{s}]\!]$.
 6:     if exploration then
 7:       exploration strategy chooses $\tilde{a}$ from $\tilde{A}(\tilde{s})$
 8:     else
 9:       $\tilde{a} = \arg\max_{\tilde{a} \in \tilde{A}(\tilde{s})} \tilde{Q}(\tilde{s}, \tilde{a})$
10:     end if
11:     take random action $a \in [\![\tilde{s}, \tilde{a}]\!]^s$
12:     observe new state $s'$ and reward $r$
13:     find new abstract state: $\tilde{s}' \in \tilde{S}(C)$
        for which $s' \in [\![\tilde{s}']\!]$.
14:     $\tilde{Q}(\tilde{s}, \tilde{a}) = \tilde{Q}(\tilde{s}, \tilde{a}) +$
        $\alpha(r + \gamma \max_{\tilde{a}' \in \tilde{A}(\tilde{s}')} \tilde{Q}(\tilde{s}', \tilde{a}') - \tilde{Q}(\tilde{s}, \tilde{a}))$.
15:   end while
16: end for
```

Consider the CARCASS $\tilde{C}_1$ (with $BK = \emptyset$ and a standard ordering) from figure 1 and that the current state s is $\{on(a,b), on(b,f), on(c,f), on(d,f), cl(a), cl(c), cl(d)\}$. This state belongs to $\tilde{s}_4$. Assume that the Q-value for abstract action $\tilde{a}_{4,2} = move(A, C)$ is the highest and we choose this one to execute. State s belongs to abstract state $\tilde{s}_4$ because it is the first state in the ordering that has s as its model, i.e. $s \vdash_\theta \tilde{s}_4$. This proof generates two possible substitutions for the variables A and C; $\theta_1 = \{A/a, C/c\}$ and $\theta_2 = \{A/a, C/d\}$. Under the given abstraction, we cannot distinguish between them and can take a random one. For example, we can take the action $move(a, c)$. After performing this action, we arrive in a new (abstract) state and receive a reward. Now we can update the Q-value of $(\tilde{s}_4, move(A, C))$ by using the Q-learning update rule. The algorithm is depicted in algorithm 1.

After learning the abstract Q-function $Q(\tilde{s}, \tilde{a})$ for all $\tilde{s} \in \tilde{S}(C)$ and all $\tilde{a} \in \tilde{A}(\tilde{s})$, we can deduce a policy $\{\tilde{s} \to \tilde{a} \mid \tilde{s} \in \tilde{S}(C) \wedge \tilde{a} \in \tilde{A}(\tilde{s}) \wedge \tilde{Q}(\tilde{s}, \tilde{a}) = max_{\tilde{a}' \in \tilde{A}(\tilde{s})} \tilde{Q}(\tilde{s}, \tilde{a}')\}$.

4 Model-based RL for RMDPs

Because the states in a CARCASS $C(M)$ partition the states of an RMDP M , it essentially defines a new discrete (PO)MDP having state set $\{[\![\tilde{s}_1]\!], \ldots, [\![\tilde{s}_n]\!]\}$. This means that we can learn a transition and reward model for this new state space. This model can be used for more efficient value function learning. The estimated probabilities and associated rewards for transitions between abstract states can be calculated from the following counters that are updated after each action:

$C^{\tilde{a}}_{\tilde{s}\tilde{t}}$ is the number of transitions from state $\tilde{s}$ to state $\tilde{t}$ after executing the action $\tilde{a}$.

$C^{\tilde{a}}_{\tilde{s}}$ is the number of times the agent has executed the abstract action $\tilde{a}$ in state $\tilde{s}$.

$R^{\tilde{a}}_{\tilde{s}\tilde{t}}$ is the sum of rewards received by the agent by executing the action $\tilde{a}$ in state $\tilde{s}$ and making a transition to state $\tilde{t}$

With these quantities we can calculate an approximate transition and reward model of the underlying RMDP:

$$\tilde{P}_{\tilde{s}\tilde{t}}(\tilde{a}) \leftarrow \frac{C^{\tilde{a}}_{\tilde{s}\tilde{t}}}{C^{\tilde{a}}_{\tilde{s}}} \qquad \tilde{R}_{\tilde{s}\tilde{t}}(\tilde{a}) \leftarrow \frac{R^{\tilde{a}}_{\tilde{s}\tilde{t}}}{C^{\tilde{a}}_{\tilde{s}\tilde{t}}} \qquad (1)$$

In algorithm 2 the prioritized sweeping (PS) method for updating the Q-values is shown. When using PS, line 14 in algorithm 1 is replaced by a call to algorithm 2. Whereas Q-learning only updates the Q-values along the experienced trace, PS updates many Q-values throughout the state-action space. By using the transition model updates for one state can be propagated to other states that have an action leading to that state. A *priority queue* (PQ) orders states by the magnitude of the change if states would be updated and updates are performed following this order.

Algorithm 2 Prioritized Sweeping (Wiering's version (Wiering, 1999))

```
Require: $\tilde{s}$ is the most recently visited state
 1: for all $\tilde{a} \in \tilde{A}(\tilde{s})$ do
 2:   $\tilde{Q}(\tilde{s},\tilde{a}) \leftarrow \sum_{\tilde{t}} P_{\tilde{s}\tilde{t}}(\tilde{a})(R_{\tilde{s}\tilde{t}}(\tilde{a}) + \gamma V(\tilde{t}))$
 3: end for
 4: Promote most recent to top of PQ
 5: n = 0
 6: while $(n < U_{\max}) \wedge (PQ \neq \emptyset)$ do
 7:   Remove top state $\tilde{s}$ from PQ
 8:   $\delta(\tilde{s}) \leftarrow 0$
 9:   for all predecessor states $\tilde{t}$ of $\tilde{s}$ do
10:     $V'(\tilde{t}) \leftarrow V(\tilde{t})$
11:     for all $\tilde{a} \in \tilde{A}(\tilde{t})$ do
12:       $\tilde{Q}(\tilde{t},\tilde{a}) \leftarrow \sum_{\tilde{u}} P_{\tilde{t}\tilde{u}}(\tilde{a})(R_{\tilde{t}\tilde{u}}(\tilde{a}) + \gamma V(\tilde{u}))$
13:     end for
14:     $V(\tilde{t}) \leftarrow \max_{\tilde{a} \in \tilde{A}(\tilde{t})} \tilde{Q}(\tilde{t},\tilde{a})$
15:     $\delta(\tilde{t}) \leftarrow \delta(\tilde{t}) + V(\tilde{t}) - V'(\tilde{t})$
16:     if $|\delta(\tilde{t})| > \epsilon$ then
17:       promote $\tilde{t}$ to priority $|\delta(\tilde{t})|$
18:     end if
19:     $n \leftarrow n + 1$
20:   end for
21: end while
22: $PQ := \emptyset$, but keep the $\delta(\tilde{t})$ values
```

In general, the decision problem is no longer *Markovian* if we use abstractions. The Markov Property holds if the output of an action does not depend in the previous actions and visited states (history), but only depends on the current state (s_i and a_i are the state and action at time i), i.e.
$T(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = T(s_{t+1} \mid s_t, a_t)$
An environment where this property does not hold is said to be *partially observable.* In general state abstraction, a condition on the abstraction that maintains the Markov property on the abstract level is the following (Dearden and Boutilier, 1997). Let $\tilde{s}$ and $\tilde{u}$ be abstract states such that $s, t \in \tilde{s}$. Then for any action a:

$$\sum_{u \in \tilde{u}} \Pr(u \mid a, s) = \sum_{u \in \tilde{u}} \Pr(u \mid a, t) \qquad (2)$$

But because a CARCASS also abstracts over actions, we need a different condition. As an illustration, see the following example[3].
Let $\tilde{s} = on(A,B) \wedge on(C,D) \wedge on(E,f) \wedge cl(A) \wedge cl(C) \wedge cl(E)$ and consider the actions $\tilde{a}_1 = move(A,C)$, $\tilde{a}_2 = move(C,A)$ and $\tilde{a}_3 = move(A,f)$. Also, let $s = \{on(a,b), on(b,c), on(c,f), on(d,e), on(e,f), on(g,f)\}$. Now we can notice the following things:
1) $\tilde{a}_1$ and $\tilde{a}_2$ both aggregate over $move(a,d)$ in s, i.e. $move(a,d) \in ([\![\tilde{s},\tilde{a}_1]\!]^s \cap [\![\tilde{s},\tilde{a}_2]\!]^s)$
2) $\tilde{a}_3$ aggregates $move(a,f)$ and $move(d,f)$, i.e. $\{move(a,f), move(d,f)\} \subseteq [\![\tilde{s},\tilde{a}_3]\!]^s$
This shows how actions are dependent on the abstractions and the specific ground state. It shows that abstract actions can have many groundings, and that the same groundings can stem from different abstract actions. This alters the way we have to look at abstract transitions, as opposed to the condition in equation 2. The condition on abstract states and actions and - therefore on a CARCASS - is the following. Let $M = \langle P, A, D, T, R \rangle$ be an RMDP. Let $\tilde{s}_i$ and $\tilde{s}_j$ be two abstract states for some $\tilde{C}(M)$. Let $s_1, s_2 \in [\![\tilde{s}_i]\!]$. The Markov property for the aggregated RMDP $\tilde{C}(M)$ is maintained if

$$\sum_{a \in [\![\tilde{s}_i,\tilde{a}]\!]^{s_1}} [\sum_{s' \in [\![\tilde{s}_j]\!]} T(s_1, a, s')] = \sum_{a \in [\![\tilde{s}_i,\tilde{a}]\!]^{s_2}} [\sum_{s' \in [\![\tilde{s}_j]\!]} T(s_2, a, s')] \qquad (3)$$

for all choices of i, j and all $\tilde{a} \in \tilde{A}(\tilde{s}_i)$. This condition ensures that the dynamic behavior will be Markov, but it does not say anything about the rewards. Rewards for abstract transitions will be averaged over all the ground transitions that are aggregated.

5 Experiments and Analysis

In this section we will show results of some experiments. Some important notions will be shown.

In the first example, we use the CARCASS $\tilde{C}_1$ of figure 1. $\tilde{C}_1$ abstracts purely from the block names; all the configurations of blocks are described as abstract states. So, although we still need as many abstract states as there are configurations, the number of states is drastically decreased (from 73 to 5).

[3]We omit the cl/1 atoms in this and following examples for shorter notation of ground states.

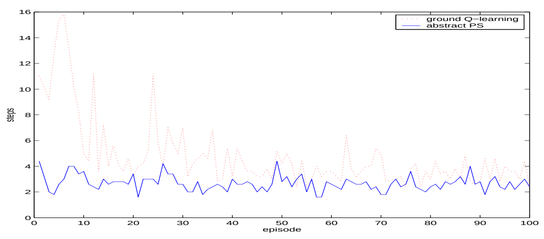

Figure 2: Q-learning on a ground state space vs. PS on an abstract state space for a four blocks world (averaged over 5 runs).

Because in this representation we just abstract from block names, the transitions are consistent with respect to the ground model and condition 3 holds. In figure 2 our model-based RL method for $\tilde{C}_1$ is compared to standard Q-learning on the ground state-space. The goal was to reach the *stack* state. After a few episodes the value function and policy are learned, whereas in ground learning, we experience high variance because it takes much more time to learn the value function for all state-action pairs. This effect becomes even more apparent when the number of blocks increases. Note that the learned policy is deterministic at the abstract level but non-deterministic at the ground level.

In figure 3 we see a more general CARCASS $\tilde{C}_2$ for blocks worlds (adopted from (Kersting and de Raedt, 2003)). This CARCASS applies to blocks worlds of arbitrary size and shows the general nature of a CARCASS.

This abstraction appears to be useful in learning how to move to a *stack* position in blocks worlds of arbitrary size. Different numbers of stacks can be discriminated and the goal state is reachable. But, due to the abstraction level, some interesting things occur. As an example, consider the first abstract state ($\tilde{s}_1$) in figure 3. It aggregates over all states where there are at least two towers of height > 2 and at least one tower of height 1. The action $move(A, C)$ moves the top of the first tower to the top of the second, whereas the action $move(A, f)$ puts it on the floor. Intuitively these actions do different things. We can show however that - due to the abstraction used - for a reinforcement learner, both are exactly the same.

All states $s \in [\![\tilde{s}_1]\!]$ are states with the following structure, e.g. a list of tower heights:

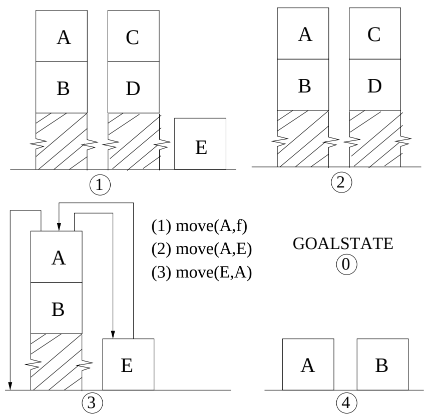

Figure 3: CARCASS $\tilde{C}_2$ for blocks worlds of arbitrary size. States are ordered by their number. The dashed blocks denote the presence of zero of more blocks. As an example, state (3) is accompanied by the available abstract actions.

$< (2 + x), (2 + y), 1, t_1 \ldots t_k >$
Now there are three different types of situations. (case 1): $x = 0$ and $\forall i = 1 \ldots k.t_i = 1$. Both actions $move(A, f)$ and $move(A, C)$ will decrease the height of the first tower to 1, so there will be only one tower left and we make a transition to $\tilde{s}_3$. (case 2): $x = 0$ and $\exists i.t_i > 1$. In this case, the first tower disappears, but two towers of height > 1 will be left and both actions result in a transition to $\tilde{s}_1$. (case 3): $x > 0$. Both actions will result in a transition to state $\tilde{s}_1$ because basically the same configuration is left, except that the first tower has it's height decreased by one.

With this we can conclude that intuitively the two actions $move(A, f)$ and $move(A, C)$ are quite different, but their transition probabilities are exactly the same and hence they will get the same Q-value and therefore both actions are equally 'good'. An optimal policy can choose either one. A similar analysis applies for state $\tilde{s}_2$ if we delete state $\tilde{s}_1$ from $\tilde{C}_2$. This shows that we have to be careful when defining abstractions, for even in a simple CARCASS such as $\tilde{C}_2$ we can easily be mistaken about the resulting learning problem.

The next example shows, by using equation 3, that $\tilde{C}_2$ is a non-Markov problem, i.e. the outcome of actions *does* depend on the history of actions. Consider a 6 blocks world: $s_1 = \{on(a, b), on(b, c), on(c, f), on(d, e), on(e, f), on(g, f)\}$ and $s_2 = \{on(a, b), on(b, f), on(c, d),$

$on(d,f), on(e,f), on(g,f)\}$. Now let $\{a_{11}, a_{12}\} \equiv [\![\tilde{s}_1, \tilde{a}_1]\!]^{s_1} = \{move(a,f), move(d,f)\}$. $\{a_{21}, a_{22}\} \equiv [\![\tilde{s}_1, \tilde{a}_1]\!]^{s_2} = \{move(a,f), move(c,f)\}$. Doing actions a_{i1} and a_{i2} in state s_i results in: $s_{11} = \{on(b,c), on(c,f), on(d,e), on(e,f), on(a,f), on(g,f)\}$, $s_{12} = \{on(a,b), on(b,c), on(c,f), on(d,f), on(e,f), on(g,f)\}$, $s_{21} = \{on(c,d), on(d,f), on(a,f), on(b,f), on(e,f), on(g,f)\}$ and $s_{22} = \{on(a,b), on(b,f)\ on(c,f), on(d,f), on(e,f), on(g,f)\}$ So, $s_{11} \in [\![\tilde{s}_1]\!]$ and $\{s_{12}, s_{21}, s_{22}\} \subseteq [\![\tilde{s}_3]\!]$. And now, by using equation 3, we can see that $T(s_1, a_{12}, s_{12}) \neq T(s_2, a_{21}, s_{21}) + T(s_2, a_{22}, s_{22})$ Which amounts to $0.5 \neq 1.0$.

Experiments using $\widetilde{C}_2$ (500 episodes, PS, $\gamma = 0.9$) resulted in the *stack-unstack* policy $\{\tilde{s}_1 \rightarrow move(A,f)(/move(A,C)), \tilde{s}_2 \rightarrow move(A,f), \tilde{s}_3 \rightarrow move(E,A), \tilde{s}_4 \rightarrow move(A,B)\}$, see also (Kersting and de Raedt, 2003)

We have argued that the given abstraction generates a non-Markovian decision problem. Learning methods such as PS and Q-learning were developed for Markovian problems. Both use only one-step look-aheads and do not cope well with these kind of problems. Using *eligibility traces* in RL, such as in $Q(\lambda)$ (Sutton and Barto, 1998) can result in better learning and CARCASS's can easily be extended with that. In the general case there are two solutions. One is a careful construction of abstractions, preferably automatically, by avoiding the introduction of partial observability. Equation 3 can be used as a guideline, similar to other work in state abstraction (Dearden and Boutilier, 1997). The other is by defining more principled methods for dealing with partial observability, i.e. by means of (probabilistic) belief states or memory to store past states.

We also performed some experiments with the game Tic-Tac-Toe. The size of the ground state space is around 6000, but there are many patterns on the board that can be described compactly in a CARCASS. As background knowledge we provide relations to describe *lines*, *winning* moves, *fork positions*, etc. For example, in an empty board the learner can move to a *corner*, *center* or *midline* square. Depending on the specific CARCASS, the compact state-action space contains approximately only $40-60$ state-action pairs (e.g. Q-values) and so learning is reasonably fast, even in case of playing against a reasonable, stochastic opponent.

6 Related Work

Very recently, interest has grown in using richer, more powerful, representation languages for MDPs. Roughly, two types of methods exist: one in which the transition function and reward functions are known (in abstract form) and one in which they are not. It is especially the last one that we are concerned with in this paper. We will mention some of them.

Some model-based methods have been proposed in the past few years. Boutilier et al. were first in proposing a relational representation for MDPs (Boutilier et al., 2001) using the *Situation Calculus* as representation language. Recently, Yoon et al. (Yoon et al., 2002) introduced a method for upgrading abstract policies for small RMDPs to larger, similar problems. Fern et al. extended the method by introducing *approximated policy iteration* (Fern et al., 2003). Both methods use a *description logics* as representation language.

In model-free approaches, we do not assume knowledge of either the transition nor reward function. From the few existing methods for *relational* RL, the majority focuses on learning a relational value function approximation. Early work dealt with batch learning from sampled states and actions (Dzeroski et al., 1998; Lecoeuche, 2001) whereas more recent work induces *relational regression trees* in an online fashion (Driessens et al., 2001). These methods combine learning the representation and the value function.

More recent work upgrades statistical learning methods originally developed for propositional or attribute-value representations to the relational case. Examples are *nearest-neighbor* techniques (Driessens and Ramon, 2003) and *kernel* methods (Gärtner et al., 2003).

Most model-free methods focus on Q-function approximation. By noticing that the underlying model for these functions is an RMDP, abstract state spaces can be defined and value functions can be learned for that. RQ learning was recently introduced (Morales, 2003) for Q-learning on separated state and action spaces. Van Otterlo (Otterlo, 2003) introduced abstract state-action spaces and Q-function learning using a learned transition model. Independently of this, Kersting and de Raedt (Kersting and de Raedt, 2003) introduced *Logical* MDPs which are very similar to the the first part in this paper.

7 Conclusions

We have presented a new model for relational MDPs and we have defined a new representational tool - the CARCASS - for RL in relational domains. A CARCASS allows for abstract (and declarative) specification of a state-action space. Learned policies can be very comprehensible and general. By focusing on a relational abstraction of the MDP model instead of a relational abstraction of the Q-function (as for example in (Driessens et al., 2001)), we can relate to existing results in the MDP literature and connect it with to various existing methods for solving MDPs.

Although we have shown the usefulness of the approach in this paper, we also highlighted some problems. In general, we will have to deal with *partial observability* in a principled manner. Criteria as discussed in section 4 can be used to characterize the *most general Markovian abstraction level*. In principle, this could be the ground level where we lose the advantages of abstraction. Furthermore, the ordering on states is a practical, yet maybe not the best method for representing partitions. Replacing it by an unordered, possibly overlapping, partition could increase the generality of the approach and allow for easier specification.

Our further research will explore the practical and theoretical consequences of using relational abstractions for MDPs. This includes convergence issues, model-based extensions and larger domains. Furthermore, we are investigating methods for learning the state-space representation. Finally, a long term goal is closing the gap between RL and programming languages for logic-based agents (Otterlo et al., 2003).

References

F. Bergadano and D. Gunetti. 1995. *Inductive Logic Programming: From Machine Learning to Software Engineering.* MIT Press.

C. Boutilier, R. Reiter, and B. Price. 2001. Symbolic dynamic programming for first-order MDP's. In Bernhard Nebel, editor, *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 690–697, San Francisco, CA, August 4–10. Morgan Kaufmann Publishers, Inc.

R. Dearden and C. Boutilier. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1–2):219–283.

K. Driessens and J. Ramon. 2003. Relational instance based regression for relational reinforcement learning. In *Proc. of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC.*

K. Driessens, J. Ramon, and H. Blockeel. 2001. Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In L. De Raedt and P. Flach, editors, *Proc. of ECML - European Conference on Machine Learning*, volume 2167 of *LNAI*, pages 97–108. Springer-Verlag.

S. Dzeroski, L. De Raedt, and H. Blockeel. 1998. Relational reinforcement learning. In J. Shavlik, editor, *Proc. of the 15th International Conference on Machine Learning (ICML'98)*, pages 136–143. Morgan Kaufmann.

A. Fern, S. Yoon, and R. Givan. 2003. Approximate policy iteration with a policy language bias. In *Proc. of the Neural Information Processing Conference (NIPS'03).*

T. Gärtner, K. Driessens, and J. Ramon. 2003. Graph kernels and gaussian processes for relational reinforcement learning. In *Proc. of the International Conference on Inductive Logic Programming (ILP'03).*

K. Kersting and L. de Raedt. 2003. Logical Markov decision programs. In *Proc. of the IJCAI'03 Workshop on Learning Statistical Models of Relational Data.*

R. Lecoeuche. 2001. Learning optimal dialogue management rules by using reinforcement learning and inductive logic programming. In *Proc. of the North American Chapter of the Association for Computational Linguistics (NAACL)*, Pittsburgh, June.

E.F. Morales. 2003. Scaling up reinforcement learning with a relational representation. In *Proc. of the Workshop on Adaptability in Multi-Agent Systems at AORC'03, Sydney.*

M. van Otterlo, M.A. Wiering, M. Dastani, and J.-J.Ch. Meyer. 2003. A characterization of sapient agents. In *Proc. of the International Conference on the Integration of Knowledge Intensive Multi-Agent Systems KIMAS'03.*

M. van Otterlo. 2002. Relational representations in reinforcement learning: Review and open problems. In E. de Jong and T. Oates, editors, *Proc. of the ICML'02 Workshop on Development of Representations.*

M. van Otterlo. 2003. Efficient reinforcement learning using relational aggregation. In *Proc. of the Sixth European Workshop on Reinforcement Learning, Nancy, France (EWRL-6).*

R.S. Sutton and A.G. Barto. 1998. *Reinforcement Learning: an Introduction.* The MIT Press, Cambridge.

M.A. Wiering. 1999. *Explorations in Efficient Reinforcement Learning.* Ph.D. thesis, Faculteit der Wiskunde, Informatica, Natuurkunde en Sterrenkunde, Universiteit van Amsterdam.

S. Yoon, A. Fern, and R. Givan. 2002. Inductive policy selection for first-order MDPs. In *Proc. of the International Conference on Uncertainty in Artificial Intelligence (UAI'02).*