# Solving Relational MDPs with First-Order Machine Learning*†

**Mausam** and **Daniel S. Weld**
Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
*{mausam,weld}@cs.washington.edu*

**Abstract**

We present a new formulation of Relational Markov Decision Processes (RMDPs) which is simpler than the situation-calculus approach of Boutilier, Reiter and Price. In addition, we describe our initial efforts developing a novel, machine-learning based method for computing an RMDP's policy. Our technique instantiates the RMDP into a number of propositional MDPs, which are then solved for their value functions. First-order regression techniques are then used to learn a value function for the complete RMDP. This value function may then be used to produce a policy for huge decision-theoretic planning problems, outputting compact solutions without actually requiring explicit state space enumeration. Finally, we extend our RMDP formalism to cover the case of a dynamic universe, i.e. in which action effects may create new objects or destroy existing ones.

## Background

A Markov Decision Process is a tuple <$S, A, T, \Re$>, where

- $S$ is the set of states.
- $A$ is the set of actions.
- $T$ is the transition function $A \times S \times S \rightarrow [0,1]$ which takes an action, the current state and the next state and gives the probability of this transition.
- $\Re$ is the reward model which is a mapping from a state to a real number. Intuitively, $\Re(s)$ is the reward the agent would get when it reaches a particular state $s$.

We consider MDPs for which the objective is to maximise the expected discounted reward gathered by the agent acting over infinite time. Thus we also include a $\gamma \in [0,1)$ as the discount factor. Since we are in the fully observable case, we need to output a policy $\pi : S \rightarrow A$ which maximises our objective function. Note that such an MDP satisfies the Bellman backup equation given by $V^*(s) = \Re(s) + \max_{a \in A} \{\gamma [\sum_{s' \in S} T(a,s,s')V^*(s')] - cost(a)\}$.

Here $cost$ represents the cost of performing an action and $V^*(s)$ is defined as the maximum expected discounted reward accumulated starting from a given state $s$.

In many cases, it is possible to factor the MDP. In this, a state is defined in terms of a set of state variables ($X = \{X_1, X_2, \ldots X_n\}$), which are often Boolean. Thus one assignment of values to state variables represents one state. Assuming Boolean state variables, one can see that $|S| = 2^{|X|}$. In this framework, the transition function for each action is best described using a DBN and even the reward model may be described using the state variables.

Considerable work has been done in developing methods for solving these propositional planning problems under uncertainty, especially assuming full observability. Most approaches have used dynamic programming of the Bellman equation in either the value iteration or policy iteration framework. Some have used heuristics to reduce the computational requirements. In the case where the states are non-factored, Bonet and Geffner (Bonet & Geffner 2000) use heuristic search over the state space. However, in the propositionally factored case, SPUDD (Hoey *et al.* 1999) uses algebraic decision diagrams to do symbolic reasoning over the state space. This method proves to be fairly fast on many reasonable sized problems. Feng and Hansen (Feng & Hansen 2002) observe that the information of an initial state could be utilised to speed up the whole procedure. They adapt SPUDD to add state reachability information by doing alternative rounds of dynamic programming and reachable state space expansion, and achieve much better results. Alternatively, Koller and Parr (Guestrin, Koller, & Parr 2001; Koller & Parr 2000; 1999) factor the value function as linear combination of small basis functions and then solve the MDP approximately by doing closed form computations.

However all this is in the realm of propositional planning that is defined in terms of ground actions and ground state variables. But, in real life, objects are divided into various classes. In a standard (propositional) MDP, one would have to specify each ground action and state variable separately. If we wish to succinctly represent such domains, we would have to consider classes of objects, and define state variables and actions over classes (or ordered tuples of classes), and provide a list of ground objects separately. To expand such a description into actual propositional planning domain we would have to parametrise all these state variables and actions over all possible legal combinations from the list of objects.

*This work is supported by ONR grant N00014-02-1-0932 and NASA grant NAG 2-1538

†An extended version also submitted to the Workshop on Planning under Uncertainty and Incomplete Information, ICAPS'03.

This idea, termed as Relational MDPs (RMDPs), was introduced in a seminal paper (Boutilier, Reiter, & Price 2001) in the context of the situation calculus (we provide a new and simpler formulation of RMDPs in the following section). The standard MDP solvers can't even take off in the expanded RMDP problems with even a small number of domain objects, as the number of state variables grow with number of domain objects fairly fast. They try to utilise the fact that all objects of the same class behave similarly so that rather than taking each object separately we can deal directly with object variables which could be parametrised appropriately in the actual problem. They learn the value function and the policy in the symbolic abstract form independent of the number of domain objects by partitioning the state space based on certain properties and assigning a value to each partition. Although their procedure is sound and elegant, they could not generate a working implementation because of the need to use first order theorem provers to prune the number of partitions as they were blowing up very fast. On the whole their system was fairly slow and they could only show results of first iteration with one time rewards after some hand pruning. More recently, Guestrin and Koller (Guestrin & Koller 2002) show a way to solve RMDPs by considering a class-based, approximate value function and solve it using linear programming combined with sampling over worlds.

## Relational Markov Decision Process

Let us extend MDPs with full observability to RMDPs with full observability as follows:

**Definition (RMDP):** We formalise a relational markov decision process as a tuple <$C, F, A, D, T, \Re$>, where $C, F$ and $A$ are all sets of relational schemata. In particular,

- $C$ is a set of classes denoting the different possible types of a ground object.
- $F$ is the set of fluent schemata. Each fluent $f \in F$ has arity $\alpha(f)$ and we assume typed logic i.e. with each fluent $f$ is associated a function $t_f : \{1, 2, ..., \alpha(f)\} \rightarrow C$. This function represents the types of different arguments for $f$ to be valid.
- $A$ is a set of action schemata and as with the elements of $F$, there is an associated arity and a type function $t_a$ for every $a \in A$. Also the cost of each action is a positive real number.
- $D$ represents a set of domain objects. With each object from the set $D$ is associated a single $type$ from $C$.
- $T$ is a transition function which represents the probabilities of transition between different states (we will discuss what comprises a state, shortly ).
- Finally $\Re$ is the reward model. For simplicity, we consider the model as a mapping from the set of states to real numbers. However, we could handle a more complex model which associates a real value with every tuple $(currentstate, nextstate, action)$

**Example:** Following the example of (Boutilier, Reiter, & Price 2001) let us consider a domain in which there are boxes in different cities and the goal is to bring one box into Paris. There are trucks which help in this transportation. To determine one state we would have to know whether a box is in a city (Bin), whether a truck is in a city (Tin) and whether a box is on a truck (On). The actions are unloading a box from the truck, loading a box onto the truck and driving the truck from a city to another. Let us formally define this RMDP.

- $C$ : {Box, Truck, City}
- $F$ : {Bin(Box,City), On(Box, Truck), Tin(Truck, City)}
- $A$ : {Unload(Box,Truck,City), Load(Box, City, Truck), Drive(Truck,City,City)}
- $\Re : if\ \exists b\ Bin(b, Paris)\ then\ 10\ else\ 0.$
- Any number of boxes, trucks and cities will give one possible $D$.
- The transition function $T$ is defined in detail in the following text.

We now define the set ($F_{D'}$) of all possible fluent tuples instantiated with elements of domain $D'$.

$F_{D'} = \{f(d_1, d_2, \ldots d_{\alpha(f)}) |\ f \in F,\ d_i \in D',\ t_f(i) = type(d_i)\}$.

If we expand the RMDP into a factored MDP then the state variables in the MDP are elements of $F_D$ - the set of possible tuples comprising the fluent relations formed by instantiating the schemata in $F$ with objects in $D$. Hence, the state space $S$ of the problem is $\wp(F_D)$, the power set of $F_D$. We can similarly define the possible actions one can execute over some set of domain objects $D'$ as $A_{D'} = \{a(d_1, d_2, ..., d_{\alpha(a)}\} |\ a \in A,\ d_i \in D',\ t_a(i) = type(d_i)\}$.

The transition function $T$ is in general a mapping from $S \times S \times A_D \rightarrow [0, 1]$. Note that the state space is extremely large and thus specifying a general transition function is impractical. An interesting restricted form is a relationally factored version of the transition function.

**Assumption 1:** We assume that an action can only affect relational fluents instantiated over its parameters.

This is a reasonable assumption for many situations since one can add an arbitrary (finite) number of parameters to an action. Thus, the assumption is akin to ruling out universally quantified effects. In such a case, we can achieve a compact specification of the transition function.

### Compact specification of transition function

Let us consider an action $a^* = a(d_1, d_2, ..., d_{\alpha(a)})$ where $a \in A$ and $d_i \in D$. Define $D^* = \cup_{i=1}^{\alpha(a)} \{d_i\}$. As assumed, the action $a^*$ can affect only $d_i$'s, i.e. it affects the fluent tuples instantiated only by domain objects from $D$. Hence, the transition function associated with $a^*$can be thought as $T_{a^*} = \wp(F_{D^*}) \times \wp(F_{D^*}) \rightarrow [0, 1]$.

We can further reduce the specification in the problems where one can assume that the value of a relational fluent in the new state is independently modified by an action irrespective of the value of other fluents in the new state i.e. depends only on the previous state. In such a case $T_{a^*} = \wp(F_{D^*}) \times F_{D^*} \rightarrow [0, 1]$.

```
Action : Unload(box,truck,city)
Preconditions: Tin(truck,city),
               On(box,truck)
Effects: Bin(box,city)∧
         ¬On(box,truck)  p = 0.8
Action : Load(box,city,truck)
Preconditions : Bin(box,city),
                Tin(truck,city)
Effects: On(box,truck)∧
         ¬Bin(box,city)  p = 0.8
Action : Drive(truck,city1,city2)
Precondition : Tin(truck,city1)
Effects: Tin(truck,city2)∧
         ¬Tin(truck,city1)  p = 0.9
```

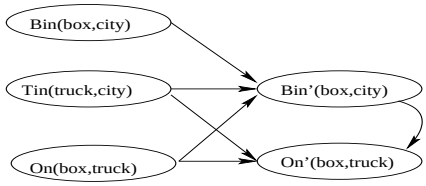
Figure 1: Transition function for actions in Probabilistic Strips representation.

Finally if we assume that each action template, $a$, behaves similarly with all the similar objects (satisfying same relational fluents), then for each action we can specify this transition function as a template and we can instantiate the parameters with different domain objects to get the exact probability of a particular transition.

**Example (contd):** Following the previous example, the figure 1, shows the transition function in Probabilistic Strips format (Boutilier, Dean, & Hanks 1999). Note that in our example, load and unload succeed with probability 0.8 and drive succeeds with probability 0.9. Moreover, all the variables that have not been mentioned are assumed unchanged. However to take advantage of the DBN representation (Dean & Kanazawa 1989) we could instead create a relational DBN representation where the state variables would be these parametrised relations which would be causes of other relational state variables. But recall that in standard DBN representations one must explicitly represent the causal relationship of each new state variable. However, in our case, doing this would greatly increase the size of the representation. So we adopt a DBN representation with an implicit persistence property which means that all the new variables that have not been mentioned remain unchanged. The transition function of $Unload(box, truck, city)$ as an example of this relational DBN with persistence is shown in figure 2.

As we consider full observability in our model, we assume that after each action execution the agent knows the new state achieved. To us, that means that after executing action $a^*$, the agent knows the value of each fluent from the set $F_{D^*}$.

The solution of such an RMDP is similar to that of the MDP i.e. to find a policy ($\pi : S \rightarrow A_D$) which maximises the expected discounted reward over an infinite horizon. We see that the Bellman backup equations can be inherited from the MDP. The optimal value function $V^*$is defined as: $V^*(s) = \Re(s) + \max_{a \in A_D} \{\gamma \left[\sum_{s' \in S} T_a(s, s') V^*(s')\right] - cost(a)\}$




| Tin | On | Bin | Bin' |
|---|---|---|---|
| F | F | F | 0 |
| F | F | T | 1 |
| F | T | F | 0 |
| T | F | F | 0 |
| T | F | T | 1 |
| T | T | F | 0.8 |

| Tin | On | Bin' | On' |
|---|---|---|---|
| F | F | X | 0 |
| F | T | X | 1 |
| T | F | X | 0 |
| T | T | T | 0 |
| T | T | F | 1 |

Figure 2: Relational DBN representation of transition function of Unload(box,truck,city). Note that all unmentioned terms (eg. Tin'(truck,city) ) will remain unchanged. Note that both Bin(box,city) and On(box, truck) can't be true at the same time.

## Solving RMDPs by learning the value function

In this section, we describe our approach to learn the first order value function for the RMDPs. Notice that it is straightforward to use this value function to generate the policy.

**Assumption 2:** We assume that the RMDP reward model is a piecewise constant function which defines a partition over state space such that each equivalence class has the same value. We also assume that each equivalence class can be defined as a quantified first order logic expression.

That is we will not handle rewards which are, for example, proportional to the number of objects satisfying certain constraints. Where (Boutilier, Reiter, & Price 2001) used deductive reasoning in generating the first order value function, we use inductive learning techniques to do the same.

1. We first expand the RMDPs with an extremely small number of objects into a ground MDP.
2. We then use a state of the art MDP solver to compute the value function of this small MDP.
3. We repeat the above two steps to generate a suitable number of training examples.
4. We now apply learning techniques using this data to generate a value function in the symbolic form. Specifically, we use a learner which generates first order regression trees (decision trees with internal nodes having quantified logic expressions and leaves as numeric values).

An example of such a value function is shown in Figure 3. This figure can be read as follows. If there is a box in Paris the value is 10, else if there is a box on some truck and that truck is in Paris then value is 7. If that truck is elsewhere then the value is 5 and so on.

Since we wish to learn a real-valued value function, we require a learning technique which ascribes numeric values (rather than a symbolic classifications) to a partition of the state space. A variant of inductive logic programming called structural regression trees (SRT) (Kramer 1996) is tailor-made for our purposes. SRT builds a sequence of increasingly complex trees (by gradually decreasing the minimum

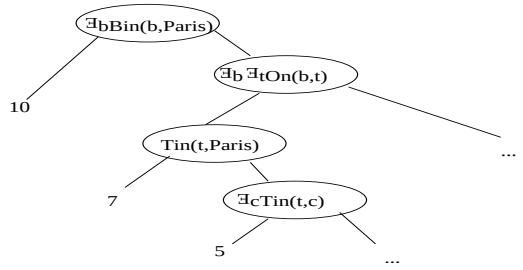



Figure 3: A value function represented as a first order regression tree. The leaf nodes are the values of the partitions. Note that all the left branches are true branches.

coverage parameter) and then chooses the best according to a minimum description length (MDL) heuristic (Rissanen 1978).

Each regression tree is grown in a manner similar to the top-down induction of a decision tree. A partition is split by considering conjunctions of literals and choosing the conjunction whose split most lowers the sum of squared differences (equation 1). For example, a set of instances $I$ might be split by a conjunction $\Gamma$ into a set, $I_1$ that satisfies $\Gamma$ and a set, $I_2$ that does not. If $\overline{y_i}$ denotes the mean of the elements $(y_{i,j})$ of $I_i$, then the sum of squared differences is:

$$\sum_{i=1}^{2}\sum_{j=1}^{|I_i|}(y_{i,j}-\overline{y_i})^2 \quad (1)$$

When the stopping criterion terminates tree growth, the tree may be used to predict the value for an arbitrary state by applying the tests at each node in turn until a leaf, $I_1$, is reached. The value assigned to the state is simply the mean value of the leaf: $\overline{y_i}$.

The major concerns in this approach are whether the assumption 2 is a good bias for learning; will the learner converge and is our method scalable. Although we don't have direct answers to these questions, our initial efforts with hand-implementation suggest that the learner will converge.

### Implementation status

We tested the algorithm by hand-executing several iterations on a small number of examples and the results seem extremely positive. This has led us to start with the implementation of the system. We use SPUDD (Hoey *et al.* 1999) as the MDP solver. We are currently modifying C4.5 (Quinlan 1993), a decision tree learner (which is written in $C$) to learn first order regression trees in the same fashion as SRT.

## Dynamic Object Relational Markov Decision Process

Consider a factory domain, where lots of widgets are being continually produced by various mills. Periodically, we need to pack and ship them appropriately. We can think of a $produce$ action that creates a new widget and a $ship$ action that takes some already produced widgets out of the system. In order to model such a domain, we need to formalise action effects that change the set of objects $D$.

Note that such a system can't be modelled by traditional MDPs as the number of domain objects, and hence the number of states is unbounded. Even if there were a bound and one added a new attribute $alive$ which told whether an object is still in system or not, this would create the problem of having a large number of objects in the working set from the very beginning and since the size of the planning problem grows unbelievably large with the number of domain objects, it would be very slow. We could try to model this with RMDPs in such a way, however, similar work in CSPs (Mittal & Falkenhainer 1990) suggests that it is faster to handle the dynamic creation of objects differently. Hence, we propose dynamic object relational MDPs (DORMDPs) as a model that caters to dynamic creation of an unbounded number of objects.

There are different possibilities for the dynamic universes in terms of the number of objects that could be created as a result of an action execution:

- **Unbounded Object Creation:** The most generic model would allow for an unbounded, variable number of new objects being created as a result of an action.
- **Bounded Object Creation:** An action producing variable number of objects subject to a maximum value.
- **Constant Object Creation:** An action producing only constant number of new objects.
- **Single Object Creation:** At most one new object being created per action execution.

For simplicity of presentation, we consider the last case in some detail. Note that, the following model can be easily extended to deal with any of above listed cases. For example, to model the first two cases, we could take the number of objects created as a probability distribution based on the current state.

**Definition (DORMDP):** A DORMDP is a tuple $<C, F, A, D, T, \Re>$ whose elements are the same as those of an RMDP except that $D$ is infinite.

The state space $S$ is $\wp(F_D)$ but since $D$ is infinite, the set of relational fluents defined over $D$ is also infinite, hence $S$ is infinite. In particular, $D$ is $D_0 \cup [\cup_{i=1}^{|C|}\{d_{i1}, d_{i2}, ...\}]$ where $D_0$ is the initial set of objects. We can shrink $D$ by having the $d_{ij}$'s only for those types $(c_i)$ for which some action creates an object of that type. Moreover, we also define two new functions over each action $pc$: $A \times S \rightarrow [0,1]$ which denotes the probability that action $A$ produces a new object in the state $S$ and $tc : A \rightarrow C$ which denotes the class of the object created.

We maintain Assumption 1, in which we assume that relational fluents are independent and that an action affects only fluents instantiated by action parameters and the new object created. In this case, the transition function can be defined as follows:

Let the action being considered be $a^*$= $a(d_1, d_2, ..., d_{\alpha(a)})$ such that $a^* \in A_D$ and $a \in A$. Let $D^*$ be $\cup_{i=1}^{\alpha(a)} \{d_i\}$. Moreover, let $D'$=$D^* \cup \{d_{new}\}$ where $d_{new}$

is the new object created such that $type(d_{new}) = tc(a)$. Also, note there would be new relational fluents created (as a result of object creation) whose truth values need to be found out. The total set of relational fluents affected would be $F_{D'}$. Then our transition function for the action $a^*$ is $T_{a^*}$: $\wp(F_D*)\times F_{D'} \rightarrow [0,1]$.

Finally the observation model for full observability requires the agent to observe whether the new object was created or not and also the truth values of all the relational fluents from the set $F_{D'}$, if new object was created and $F_{D^*}$, if it was not created. The definitions of policy and reward model *etc.* can be inherited from the RMDP definition. And as usual, the goal of such a planning problem is to find an optimal policy ($S \rightarrow A_D$) such that discounted sum of expected rewards is maximised over an infinite horizon.

We know that DORMDPs can't be expanded into propositional MDPs directly since the state space is infinite. So, the only way to generate a value function, in this case, seems to be generating a first order one, partitioning the state space based on attributes of the states. Note that this work is still ongoing.

## Conclusions

While our work is ongoing, we have already made the following contributions:

1. We defined Relational MDPs. Although (Boutilier, Reiter, & Price 2001) has already done this in the context of situation calculus, we believe that our formalism is more practical.
2. We presented a new solution method based on relational decision tree learning from the solution of expanded propositional MDPs.
3. We defined Dynamic Object Relational MDPs which allow one to model actions whose effects created objects.

In future, we will complete the implementation of our system and do experiments over it; these will answer our concerns on the convergence and scalability of the system. Further, we wish to relax the two assumptions we have made in the paper. For instance, we could deal with reward models which have rewards proportional to number of objects of a certain type and handle universally quantified effects. We could use First Order Regression System (FORS) (Karalic & Bratko 1997) since it has the ability to learn regression models over attributes. We can further incorporate the objective of trying to achieve goals in our framework, instead of maximising rewards. We also wish to look for improved reachability analysis and better heuristics to speed up the system. Another direction of research is including temporal duration in uncertain actions. We will also continue to work on solving dynamic object RMDPs.

## Acknowledgements

We are thankful to Pedro Domingos for his inputs at various stages of this research. We also thank Kate Deibel, Lin Liao, Don Patterson and Sumit Sanghai for giving useful comments on an earlier draft of the paper.

## References

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth Interational Conference on Artificial Intelligence Planning and Scheduling*, 52–61. Breckenridge, CO: AAAI Press.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 690–697. Seattle, WA: Morgan Kaufmann.

Dean, T., and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

Feng, Z., and Hansen, E. 2002. Symbolic heuristic search for factored markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*. Edmonton, Canada: AAAI Press.

Guestrin, C., and Koller, D. 2002. Generalizing plans to new environments in relational mdps. Unpublished Manuscript.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Max-norm projections for factored MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 673–682.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 279–288. San Francisco, CA: Morgan Kaufmann.

Karalic, A., and Bratko, I. 1997. First order regression. *Machine Learning* 26:147–176.

Koller, D., and Parr, R. 1999. Computing factored value functions for policies in structured MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1332–1339.

Koller, D., and Parr, R. 2000. Policy iteration for factored MDPs. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 326–334.

Kramer, S. 1996. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 812–819. Cambridge, Menlo Park: AAAI Press.

Mittal, S., and Falkenhainer, B. 1990. Dynamic constraint satisfaction problems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 25–32.

Quinlan, J. R. 1993. C4.5: Programs for machine learning.

Rissanen, J. 1978. Modelling by shortest data description. *Automatica* 14:465–471.