Relational Sequential Decision Making

Kaushik Roy
School of Informatics and Computing Science
Indiana University
Bloomington, Indiana

Abstract

Markov Decision Processes(MDPs) are the standard for sequential decision making. Comprehensive theory and methods have been developed to deal with solving MDPs in the propositional setting. Real world domains however are naturally represented using objects and relationships. To this effect, relational adaptations of algorithms to solve MDPs have been proposed in recent years. This paper presents a study of these techniques both in the model based and model free setting.

Background

Markov Decision Process

Markov Decision Processes(MDPs) are described by a set of discrete states S, a set of actions A, a reward function $R(s,a)$ that describes the expected immediate reward in state s when executing an action a, and a state transition function $p^a_{ss'}$ that describes the transition probability from state s to state s' under action a. For infinite horizon problems a discount factor γ is specified to trade-off between current and future reward. A policy π is a mapping from states to actions. Value functions evaluate a policy that is determines the value of executing actions with respect to that policy. The optimal value can be obtained by solving the bellman optimality equation as shown in Equation 1. Q-values are values defined over state and action pairs and the value can be obtained by maximizing over Q-values corresponding to each action i.e $V^*(s) = max_a(Q(s,a))$ for a state s and all actions a executable in state s.

$$V^*(s_i) = max_a\{R(s_i,a) + \gamma \sum_j p^a_{s_i,s_j} \cdot V^*(s_j)\} \quad (1)$$

Model based methods

Model based approaches in classical RL assume full knowledge of the transition model and immediate reward function. Let's consider first a unified view of two fundamental representation ideas, i.e propositional representations and representations that group sets of states together. This grouping can be done by factoring the states into state variables and grouping together sets of states that have common variable assignments. The grouping can also be performed by providing a logical description of states and grouping together states that are logically equivalent in terms of the objects and relationships over which they are defined. For the unified view however, the manner of grouping is abstracted out. This is to show that the algorithmic details of solving MDPs do not change across different representations. Adapting to the relational setting simply requires model definitions to be provided using logical formalisms.

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Representing value functions and policies over sets of states

Consider an MDP with states S, actions A, transition model T and immediate reward model R,
Value function: The value function over sets of states is defined as a list $V = [S_i : v_i, ..., S_n : v_n]$, where each S_i is a group of states $s \in S$. It follows that the value for all states $s \in S_i$ is v_i.
Q-Value function: The Q-value function over sets of states and actions is defined as a list $Q = [S_i, a_i : q_i, ..., S_n, a_n : q_n]$, where q_i is the value of taking action a_i in the states $s \in S_i$ and each $a_i \in A$.
Policy: A policy over sets of states is defined as a list $\pi = [S_i : a_i, ..., S_n : a_n]$. It is a mapping from sets of states $s \in S_i$ to action a_i.

Operations over value structures

For any two value functions V_1 and V_2, the **Sum, Difference** and **Product** operations are defined
Sum:It is defined as a list $V_1 + V_2 = [S_i \cap S_j : v_i + v_j]$, where $S_i, v_i \in V_1$ and $S_j, v_j \in V_2$.
Difference: It is defined as a list $V_1 - V_2 = [S_i \cap S_j : v_i - v_j]$, where $S_i, v_i \in V_1$ and $S_j, v_j \in V_2$.
Product: It is defined as a list $V_1 * V_2 = [S_i \cap S_j : v_i * v_j]$, where $S_i, v_i \in V_1$ and $S_j, v_j \in V_2$.
Reduce: For a single value function V it is defined as a list $Reduce(V)$ that merges $S_j, S_k \in V$ that contain the same values ($v_j = v_k$). For Q-Value structures, the actions a_j and a_k also need to match.
Maximization: For a Q-value function Q and a set of states S, action a and q-value q, where $S, a : q \in Q$, Let H be a list that collects all sets of states S', action b and q-value q', where $S', b : q' \in Q$, such that $S \cap S' \neq \emptyset$ and $q' > q$. Let

H' be the list $Reduce(H)$. The maximization is defined as the list $maxQ = [S - H']$.

Example Let $V_1 = [\{s1, s2, s3\} : 1, \{s4, s5, s6\} : 2]$ and $V_2 = [\{s1, s6\} : 8, \{s2, s3, s4, s5\} : 9]$. The **Sum** $V_1 + V_2 = [\{s1\} : 9, \{s6\} : 10, \{s4, s5\} : 11, \{s2, s3\} : 10]$. The operations **Difference** and **Product** are carried out similarly. The **Reduce** operator applied on $V_1 + V_2$ is $Reduce(V_1 + V_2) = [\{s1\} : 9, \{s2, s3, s6\} : 10, \{s4, s5\} : 11]$.
Let $Q = [\{s1, s2, s3\}, a : 10, \{s1\}, b : 5, \{s2, s3\}, b : 15, \{s1\}, c : 15, \{s2\}, c : 15, \{s3\}, c : 7]$ and $S = \{s1, s2, s3\}, a : 10$. Therefore, $H = [\{s2, s3\}, b : 15, \{s1\}, c : 15, \{s2\}, c : 15]$. Applying **Reduce** on H the list $H' = [\{s2, s3\}, b : 15, \{s1, s2\}, c : 15]$ is obtained. Finally, **Maximization** is carried out by performing $S - H'$ resulting in $maxQ = [\{s2, s3\}, b : 15, \{s1, s2\}, c : 15]$.

Example world

Described now is an example world that will help illustrate the algorithmic details similar across all representations and show how computation is carried out over sets of states instead of over all possible states. Let the set of states in the world $S = \{s1, s2, s3, s4, s5\}$ and let the actions allowed be $A = \{a\}$. Let the discount factor $\gamma = 0.9$. Action a can lead to two outcomes, aS representing success with probability 0.3 and aF representing failure with probability 0.7 denoted as $p[aS] = 0.3$ and $p[aF] = 0.7$. State $s1$ executing action aS leads to state $s4$, represented as $T(s1, aS) = s4$. Similarly, $T(s1, aF) = s5$, $T(s2, aS) = s4$, $T(s2, aF) = s2$, $T(s3, aS) = s3$ and $T(s3, aF) = s5$. Let the initial value function reward being in state $s4$ with a value of 10, and being in state $s5$ with a value of 5, represented as $V^0 = [\{s4\} : 10, \{s5\} : 5]$.

The Regression Operator

The regression operator applied to a state s and action a results in all possible states s' that can lead to the current state s upon execution of action a, denoted by $Regr(s, a) = s'$. For the example in consideration, $Regr(s4, aS) = \{s1, s2\}$ as both $s1$ and $s2$ can lead to transition to $s4$ upon successful execution of action a (action aS).

Some additional Notations

Let $outcomes(a)$ denote the set of action outcomes a_j of action a. $Q^{s,a}$ outputs the q-value q of taking action a in state s, where $s, a : q \in Q$. Similarly V^s outputs the value v of state s, where $s : v \in V$. The operator $AddQ$ adds all the Q^{s,a_j} lists using the **Sum** operator while ignoring the actions such that the result is the Q-value $Q^{s,a}$ of action a, where each $a_j \in outcomes(a)$.

Computation of Q_{k+1} and V_{k+1} from V_k

Computation of Q_{k+1} from V_k is carried out using the steps in algorithm 1.

Example The Q_1 value for action aS can be computed using these definitions as the value of states that can lead to $s4$, multiplied by the probability of selecting action aS from $outcomes(a)$. This probability is 0.3 as already mentioned for the example in point. Thus, the Q_1 value for action aS is over states $\{s1, s2\}$ as these states can lead to state $s4$ in the value function V^0 (since aS cannot lead to state $s5$, this outcome is ignored). Therefore, Q_1 for action aS is $[\{s1, s2\}, aS : \gamma * 0.3 * 10 = 2.7]$ and similarly Q_1 for action aF is $[\{s1, s3\}, aF : \gamma * 0.7 * 5 = 3.15]$. The expected value of executing action a with probailistic outcomes aS and aF is the sum over the Q_1 values of aS and aF. Combining Q_1 for aS and Q_1 for aF using $AddQ_1$, we have Q_1 for action a as $[\{s1, s2\} \cap \{s1, s3\} : 2.7 + 3.15] = [\{s1\}, a : 5.85]$. Now consider Q_1 for another action b has already been computed as $[\{s1\}, b : 6]$. The maximization over actions a and b is given by $maxQ_1 = [\{s1\}, b : 6]$. Let $maxV_1 = [\{s1\} : 6]$ i.e. the same as the $maxQ_1$ list with the action dropped . V_1 is obtained as $V_1 = R + maxV_1$, where R is the initial reward model and is always set to the list V_0. Thus, V_{k+1} is easily obtained from Q_{k+1} and the initial reward model V_0.

Result: Computing Q_{k+1} from V_k
forall $S_i : v_i \in V_k$ **do**
  forall $a \in A$ **do**
    forall $a_j \in outcomes(a)$ **do**
      $Q_{k+1}^{Regr(S_i),a_j} = \gamma * p[a_j] * V_k^{S_i}$
    end
    $Q_{k+1}^{Regr(s_i),a} = AddQ_{k+1}$
  end
end
$Q_{k+1} = maxQ_{k+1}$

Algorithm 1: Computing Q_{k+1} from V_k

Value Iteration and Policy Iteration

Computing V_{k+1} from V_k by successive approximation as detailed in the previous subsection until convergence (or until the max difference is less than ϵ) is the value iteration algorithm. Policy iteration can also be carried out by evaluating Q_{k+1} for the actions described by a policy π and then taking the action given by $maxQ_{k+1}$.

The propositional case If every state $s \in S$ in the MDP model was in its own set and every action $a_j \in outcomes(a)$ was treated as an individual action , it is easy to see that the above algorithmic procedure reduces to computation of the bellman optimally equation given by $V^*(s) = max_{a_j}\{R(s, a_j) + \gamma \sum_j p[a_j] \cdot V^*(s')\}$, where $s \in Regr(s', a_j)$.

The Relational case

Adapting the algorithmic procedure to the relational case simply involves representing sets of states using logical formalisms. For example, instead of representing the set of birds that can't fly as $\{penguins, ostriches, ...\}$ of which there could be too many, a more compact representation would be to use a logical statement such as "All birds that cannot fly". This can be written in first order logic as, $\forall x \in B \cdot bird(x) \wedge \neg fly(x)$, where B is the set of all birds.

First Order dynamic programming algorithms

The first method that was published on exact solving of Relational MDPs is the Symbolic Dynamic Programming (SDP) approach (Boutilier, Reiter, and Price 2001). The parts of algorithm 1 that need elaboration to deal with the logical framework used in SDP, which is the Situation Calculus (SC) framework (Levesque, Pirri, and Reiter 1998) is how to perform $Regr$ over logical descriptions of sets of states. The full framework can be found in the original paper. Here, a simple example is illustrated. The domain used for illustration is called the box world or logisitics domain. This domain consists of boxes and trucks in various cities and the goal is to make sure there is at least one box in the city of Paris.

The initial value function V_0 is a list $[\exists b \cdot BoxIn(b, Paris, s) : 10, \neg\exists b \cdot BoxIn(b, Paris, s) : 0]$, where s is a situation term that can be thought of as the state. The logical formula assigned the value 10 is a set of all states s where there is at least one box in Paris and similarly, the set of states where there isn't a box in Paris is assigned the value 0.

Regression operator Let's say the action $Unload(b*, t*)$ is performed, which means to unload box $b*$ from truck $t*$. This action can result in probabilistic outcomes $UnloadS(b*, t*)$ denoting successful execution and $UnloadF(b*, t*)$ denoting unsuccessful execution with probabilities 0.9 and 0.1 respectively. $Regr(\exists b \cdot BoxIn(b, Paris, s), UnloadS(b*, t*)) = BoxOn(b*, t*, s) \wedge TruckIn(t*, Paris, s) \vee \exists b \cdot BoxIn(b, Paris, s)$. This means that the regression over states in which there is at least one box in Paris upon unloading box $b*$ from $t*$ is all states where either box $b*$ is on the truck $t*$ in Paris or there already is at least one box in Paris.

Q-Value computation Thus, the Q-value of the set of states $BoxOn(b*, t*, s) \wedge TruckIn(t*, Paris, s) \vee \exists b \cdot BoxIn(b, Paris, s)$ and action outcome $UnloadS(b*, t*)$ is calculated using the procedure outlined in algorithm 1 as $\gamma * p[UnloadS(b*, t*)] * V_0^{\exists b \cdot BoxIn(b,Paris,s)} = 0.9 * 0.9 * 10 = 8.1$. Similarly, action outcome $UnloadF(b*, t*)$ regresses to set of states $\exists b \cdot BoxIn(b, Paris, s)$ whose value is given by $0.9 * 0.1 * 10 = 0.9$. Q-value for action $Unload(b*, t*)$ is computed using the $AddQ$ operator in algorithm 1 to obtain the list $[BoxOn(b*, t*, s) \wedge TruckIn(t*, Paris, s), unload(b*, t*) : 8.1, \exists b \cdot BoxIn(b, Paris, s), unload(b*, t*) : 9]$.

Value Iteration It should now be clear that the procedure to obtain Q_{k+1}(which is maximized over all actions) from V_k for First Order descriptions is carried out using the same algorithmic procedure outlined in algorithm 1. The only difference being that sets of states are now represented using first-order logical formulas. Hence, value iteration and policy iteration can be performed in a similar manner as already described before.

Other logical formalisms Holldobler et al., developed value iteration using fluent calculus as their representation(Hölldobler and Skvortsova 2004). The most efficient implementation of first-order dynamic programming is REBEL by Kersting et al. It uses a simpler logical language, the probabilistic strips language(Kersting, Otterlo, and De Raedt 2004).

Tree structured definitions Tree structured definitions such as the tree in Figure 1 (left). can be used to represent value functions that can exploit context specific independence (CSI). The nodes of the trees are propositions that map to first order formula and the leaves are values (**?**). For example, Consider the value structure $V = [\exists x \cdot [A(x) \vee \forall y \cdot A(x) \wedge B(x) \wedge \neg A(y) : 1, \neg(\exists x \cdot [A(x) \vee \forall y \cdot A(x) \wedge B(x) \wedge \neg A(y)) : 0]$. Pushing down quantifiers in the formula $\exists x \cdot [A(x) \vee \forall y \cdot A(x) \wedge B(x) \wedge \neg A(y)$ over relevant variables, the equivalent formula $[\exists x \cdot A(x)] \vee ([\exists x \cdot A(x) \wedge B(x)] \wedge [\forall y \cdot \neg A(y)])$ is obtained. Mapping a to $\exists x \cdot A(x)$ and b to $\exists x \cdot A(x) \wedge B(x)$, the value structure expressed using these proposition mappings is $V = [a \vee (b \wedge \neg a) : 1, \neg(a \vee (b \wedge \neg a)) : 0]$. This can be represented using a tree structure as shown in Figure 1 (left). In the tree, the left child represents the true branch and right child represents the false branch. Figure 1 (right) shows CSI, when the value of the proposition b in known.

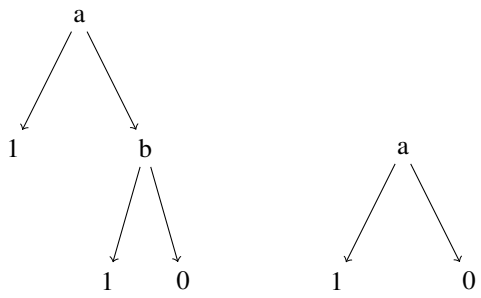

Figure 1: **Left** represents the tree structured value function and **right** illustrates CSI when the value of b is known.

Regression over tree structured definitions Regression over tree structured definitions is carried out by a procedure known as block replacement (Boutilier et al. 1995). The value tree induces an ordering over the propositions in the tree. Block replacement is carried out by appending the transition probability tree structures(PTs) of propositions appearing later in the ordering to the leaves of PTs for propositions appearing before in the ordering. Consider the PTs for propositions a, b under action A and the initial value function V_0 shown in Figure 2. The $Regr$ operator returns the states that can reach a state given by a logical formula defined over a and b under the action A. The way to read the PT in Figure 2.(left) is, if a is true, then it remains true with probability 1.0 upon execution of A. If a is false then the probability of a being true upon execution of A is 0.0. As already stated before, the value function imposes an ordering over the propositions which in this case is the order $a \prec b$. Respecting the proposition ordering imposed by V_0, the PT for proposition a is appended to the leaves of the PT for proposition b to obtain the tree in Figure 3 (the full tree i.e. the sub-tree rooted at d is not shown for brevity).

To compute the Q-Value over the regressed state $b \wedge a$ represented by the left most branch of the tree in Figure 3, upon execution of action A the state $b \wedge a$ stays unchanged (persists with probability 1.0) and therefore the Q-value is $\gamma * 1.0 * V_0^{b \wedge a} = 0.9 * 1.0 * 0.9 = 0.81$ and the value is obtained by adding immediate reward $V_0^{b \wedge a}$ which results in $0.81 + 0.9 = 1.71$. The same kind of computation is carried out for other states. Maximization over actions is performed by selecting the max value assigned per action to each regressed state under that action. Wang et al., employ block replacement, which operationally is regression over another kind of tree structured definition known as First order decision diagrams(Wang, Joshi, and Khardon 2008). Exposing propositional structure is useful because efficient propositional solvers on decision diagrams like SPUDD(Hoey et al. 1999) can be leveraged for computation.

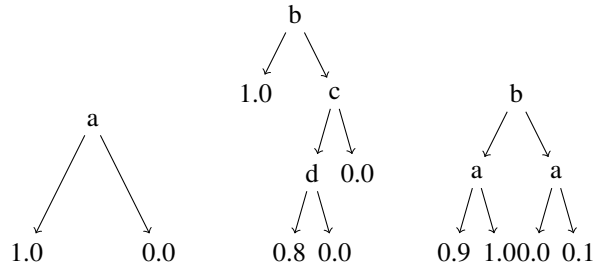

Figure 2: **Left** represents the PT for action A for proposition a, **middle** represents the PT for proposition b and **right** represents the tree structured value function over propositions a and b.

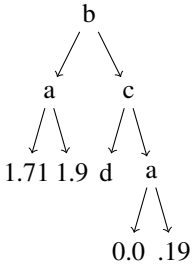

Figure 3: The states obtained after regression of the states in V_0 under action A i.e. $Regr(s, A)$, where each s is a branch of the tree V_0. The Q-values computed are shown at the leaves (The entire tree is not shown for brevity)

Approximate methods

Approximate methods mainly rely on approximate representations of the value functions when performing dynamic programming. Sanner et al., use a set of basis functions that partition the state space, expanded using the regression step to represent the value function(Sanner and Boutilier 2012). Gretton et al., don't fully perform the regression step (that is they don't perform maximization) and instead induce policy from the partially regressed state partitions(Gretton and Thiébaux 2004). Wu et al., follow a distinctly different approach by performing dynamic programming over a set of basis functions learned by fitting the bellman error of the current approximation(Wu 2007). Value functions represented using tree structures can maintain ranges of values at the leaves and provide approximation bounds in doing so(Boutilier et al. 1995).

Model free methods

Model free methods are not provided the transition and the reward model. These methods work on simulations and learn Q-values from simulation data. In the relational setting, Q-values are first-order data structures. States during simulation contain objects and relationships that hold between these objects and their attributes. Also, the actions are more complex and are over objects in the current state.

Parameter estimation - Estimating the parameters of first-order structures provided upfront

In these methods, the first-order states and actions are provided and their values are updated based on simulation data.

first-order states and actions Consider $unload(b*, t*)$ in the logistics domain, which means unload box $b*$ from truck $t*$. The key observation in this action description is that it contains no variables and is over specific objects $b*$ and $t*$. A first-order action is of the form $A = action(var_1, var_2, ..., var_n)$ which can have multiple substitutions θ^A subject to state admissibility and integrity constraints. For example, $unload(B, T)$ is a first-order action to unload a box from a truck. Again, this is subject to state admissibility and integrity constraints. For instance, You cannot unload a box from a truck in a state where no truck has any boxes on it (state admissibility) and B has to be of type box and T has to be of type truck (integrity constraints). Now, consider a state (herbrand interpretation) $s = [On(b1, t1), On(b2, t1), Tin(t1, Paris)]$. A first-order state is a list of predicates $S = [pred1(var_1, ..., var_n), ..., predN(var_1, ..., var_n)]$, that represents the set of all substitutions θ^S that satisfy the state description. For example, the first-order state for s is $[On(B, T), Tin(T, Paris)]$.

States and action structures S is a set of all first-order states and A is a set of all first-order actions. Let a list L containing first-order states and actions admissible over those states be defined as $L = [S_1 : \{A_{11}, A_{12}, ..\}, S_2 : \{A_{21}, A_{22}, ...\}, ..., S_n, \{A_{n1}, A_{n2}, ...\}]$, where each $S_i \in S$ and each $A_{ij} \in A$ is a first-order action admissible in state S_i denoted by $admissible(S_i)$. Also, for any two states S_i and S_j, $S_i \cap S_j = \emptyset$. For example, in the logistics domain the first-order state action pair $On(B, T), Tin(T, Paris) : unload(B, T)$ is a member of L for the logisitcs domain.

Transition model and Reward function For two first-order states $S_i \in S$ and $S_j \in S$, let Θ^{S_i} be a list of all substitutions (also known as ground states) $\theta_k^{S_i}$ for state S_i denoted as $\Theta^{S_i} = [\theta_1^{S_i}, \theta_2^{S_i}, ..]$. Let $\Theta^{S_i, A}$ for a first-order action A, be a list of all substitutions $\theta_k^{S_i, A}$ of the

first-order action A admissible in some state substitution $\theta_l^{S_i} \in \Theta^{S_i}$. $T(\theta_k^{S_i}, \theta_l^{S_j}, \theta_p^{S_i,A})$ is a ground transition and represents the transition probability of executing a substitute action $\theta_p^{S_i,A \in \Theta^{S_i,A}}$ in ground state $\theta_k^{S_i} \in \Theta^{S_i}$ and transitioning to ground state $\theta_l^{S_j} \in \Theta^{S_j}$, then the probability of transition from first-order state S_i to first-order states S_j upon execution of first-order action A is an aggregate over all possible ground transitions. More formally it is $\beta(\theta_p^{S_i}) * \sum_{\theta_p^{S_i,A}} \sum_{\theta_l^{S_j}} T(\theta_k^{S_i}, \theta_l^{S_j}, \theta_p^{S_i,A})$, where $beta$ is a weight to ensure the aggregation probabilities are well-defined (sum to a total of 1). Similarly, the reward model is $\beta(\theta_p^{S_i}) * \sum_{\theta_p^{S_i,A}} \sum_{\theta_l^{S_j}} R(\theta_k^{S_i}, \theta_l^{S_j}, \theta_p^{S_i,A})$, where R is the immediate reward function over ground states and actions. These models are used to simulate the domain.

Q-Value The Q-value for the list L is a function that gives a value over a first-order state,action pair $S, A \in admissible(S)$ given by $Q_L(S, A)$. This can be calculated using the standard procedure of Q-learning (Watkins and Dayan 1992), over generating samples and updating the corresponding first-order state action pair using the Q-learning update. Of course, other methods such as TD(λ) (Tesauro 1995) can also be used for learning. The key observation is that these methods chose the best first-order action for each first-order state. Therefore, the true optimal policy for any ground instantiating of the first-order states and actions may not be the one suggested by the first-order action. Intuitively, this makes sense as aggregate transition probabilities and rewards are used for learning updates, leading to a focus on getting average case best performance over the family of ground Relational MDPs. Otterlo et al., use a logical abstraction to represent first-order states and actions known as CARCASS (Van Otterlo 2004). It requires fixed first-order states and actions provided upfront and performs learning over these structures. They demonstrate learning using Q-learning and the Prioritized Sweeping algorithm (Moore and Atkeson 1993) adapted to use logical representations. Similar approaches include the Logical Markov Decision Programs (LOMDP) by Kersting et al.,(Kersting and De Raedt 2003) the difference being the language employed for the specification for the first-order structures. In the CARCASS approach, a full PROLOG style language is used where as in LOMDPs, probabilistic strips is used. Learning was performed for LOMDPs, using logical adaptations of Q-Learning and the TD(λ) algorithm. The expressive power of the language used trades of complexity of implementation and speed versus accuracy and range of queries that can be provided to the system (Probabilistic strips being a simpler less expressive representation). A third approach that is similar is the relational Q-learning(RQ) approach by Morales et al(Morales 2003). The difference in this approach is that the actions applicable is specified globally instead of for every first-order state. They use first-order relations to describe a partition over the state space into first-order states known as r-states. Every r-action is a first-order action that can be executed in an r-state if a specified precondition is met.

Learning structure

Q-RRL(Džeroski, De Raedt, and Driessens 2001) proposed by Dzeroski et al., employ a combination of Q-learning and TILDE tree(Blockeel and De Raedt 1998) learning using Inductive Logic Program to learn the Q-value as a first-order logical decision tree known as a Q-tree. At every step (after every episode), samples are generated using the current Q-tree values and a logical decision tree is induced from the samples. An example of a Q-tree can be seen in Figure 4. Walker et al., learn first-order state and actions from sam-

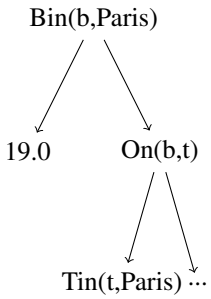

Figure 4: An example of a Q-tree for the logistics domain, left representing the true branch and right the false branch.

ples by backward induction(Walker et al. 2007). They start from the goal in the episode, the idea being to start from high reward states and generalize using Inductive Logic Programming over states with k-steps to go, known as preimages. The appeal in this approach suffers from a heuristic based calculation of Q-values of the preimages and the lack of evaluation for relational domains such as blocks world, wumpus world or the logistics domain. Learning structure can also be done by treating the value function as being probabilistically dependent on relational features. Of course, fitting regression functions is also on the basis of probability theory - however, probabilistic approaches use direct reasoning over the probabilities of dependent variables to compute agent behavior. Sanner et al., perform structure and parameter learning over an imposed naive bayes net structure and employ two methods of feature discovery - feature attribute augmentation(FAA) and feature conjunction(FC)(Sanner 2005). Attribute augmentation is carried out where the feature contains a relation of high arity, in which case the information provided by ground structures using this feature is very little (because of too many potential ground attribute instantiations). Feature conjunction is performed to check for whether or not combining to features could provide additional information about the high value states. Traditional bayes net parameter estimation and structure learning algorithms can be leveraged making this approach attractive. In relational domains however, an exhaustive search often doesn't lead to exploration of frequently visited parts of the state space (or takes prohibitively long). To address this problem, Sanner incorporates frequent item set mining to discover sets of features in frequently visited parts of the state space(Sanner 2006). Croonenborghs et al., propose a reinforcement learning agent that learns features

to predict the probability of rewards and the truth value of features in the next transition(Croonenborghs, Ramon, and Bruynooghe 2004). It can be thought of as estimating structure and parameters of a two slice dynamic bayes net.

Value function approximation using first-order features

When smooth estimates of the value function are required, the success of propositional function approximation can be extended to the relational domain by using relational features. One way to do this is to provide a set of first-order features (possibly hand-crafted) and learn the value of a state as a function of the design features using some combination of utilities assigned per feature (A linear combination for instance). Walker et al., induces a set of first-order features by sampling from a large space of features and using these as basis functions to a regression algorithm to estimate the Q-values(Walker, Shavlik, and Maclin 2004). Asgharbeygi et al., propose the Relational Temporal Difference learning algorithm that encodes the value of first-order states as a linear combination of utilities attached to each first-order feature weighted by the number of states that it applies to. Each feature is a concept predicate (for example HasO to represent if there is an 'O' in a square in Tic-tac-toe) that is supplied by a designer(Asgharbeygi, Stracuzzi, and Langley 2006). For instance, the value could be a linear combination of the number of O's and X's. Dynamic induction of first-order features could be carried out using a process similar to Q-RRL. Driessens et al., performs incremental learning of TILDE trees instead of inducing the entire tree from examples created after every sample(Ramon, Driessens, and Croonenborghs 2007). A bottleneck however, is the computation costs of incremental induction. A possible solution to this was proposed by Goetschalckx et al., where during the induction of predicates, a cost is assigned to the predicate as a function of the probability of the states that it generalizes over and their utilities(Goetschalckx and Driessens 2007).

Searching in policy space and policy gradient methods

These methods directly learn policies by starting with a policy, generating samples suggested by the policy and generating new policies subsequently until convergence. Policies are more amenable to generalization and transfer - an important part of relational learning. Muller et al., propose a policy search algorithm based on genetic algorithms that start with a population of policies and learn the right policies by evaluation against a fitness function through subsequent generations(Muller and Van Otterlo 2005). Mellor et al., perform genetic modifications over individual rules instead over the entire policy structure(Mellor 2006). Policy learning can also be performed by a sequence of functions that perform supervised learning based on Inductive Logic Programming from sampled states and actions. Fern et al., propose an approach that performs approximate policy iteration by inducing a policy structure from samples, generating new samples from the policy structure learned and induce an improved policy until convergence(Fern, Yoon, and Givan 2006). The key feature of this approach is to solve the issue of sampling to learn effectively in relational domains. They use policy rollout over a fixed horizon length instead of just a one step rollout. They avoid sampling instances of low value by performing random walks over the Relational MDP structure (which is a markov chain that can be unrolled into a graphical model)(Fern, Yoon, and Givan 2004). Policy gradient techniques optimize parametrized policies, which in the relational case pertains to parametrized logical structures. The perform gradient descent with respect to the policy parameters. The advantage of this approach is to chose a policy representation that is meaningful and can incorporate domain knowledge, thus reducing the problem to parameter estimation. Kersting et al., propose an approach that learns both the structure and parameters of the policy using gradient descent in functional space using Relational Functional Gradient Boosting, as often times fixing the logical structure beforehand may not be easy (Kersting and Driessens 2008). Domain knowledge can be incorporated into this learning paradigm as well as has been demonstrated recently by Natarajan et al.(Natarajan et al. 2012)(**?**).

The partially observable case

In a partially observable Markov Decision Process(POMDP), the states are not directly observed. The state has to be inferred from observations. This strictly subsumes an MDP when the observations are the states themselves. POMDPs in relational setting present a great opportunity to integrate graphical models, Inductive Logic Programming and utility theory in one sequential decision making framework. In propositional setting the set of observations possible at states is carefully defined and limited in number. This allows for solving them using dynamic programming approaches that run in time exponential in the size of the observation space. The exponential blowup is a concern. However, it is still finite. The additional difficulty in designing algorithms to deal with POMDPs in the relational setting involves a rich observation space that can also be defined over objects and relations in the same way as states. This important detail essentially amounts to an unbounded observation space over arbitrary instantiations. Work has been done on first-order POMDPs however, significant challenges still persist and offer an interesting direction for future research.

Acknowledgements

I would like to thank my advisor Professor Sriraam Natarajan for his support and guidance and for providing this opportunity to explore this interesting field. I would also like to thank the other committee members Professor Vibhav Gogate and Professor Nicholas Ruozzi for agreeing to review this work.

References

Asgharbeygi, N.; Stracuzzi, D.; and Langley, P. 2006. Relational temporal difference learning. In *Proceedings of the 23rd international conference on Machine learning*, 49–56. ACM.

Blockeel, H., and De Raedt, L. 1998. Top-down induction of first-order logical decision trees. *Artificial intelligence* 101(1-2):285–297.

Boutilier, C.; Dearden, R.; Goldszmidt, M.; et al. 1995. Exploiting structure in policy construction. In *IJCAI*, volume 14, 1104–1113.

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order mdps. In *IJCAI*, volume 1, 690–700.

Croonenborghs, T.; Ramon, J.; and Bruynooghe, M. 2004. Towards informed reinforcement learning. In *Proceedings of the ICML'04 workshop on relational reinforcement learning*, 21–26.

Džeroski, S.; De Raedt, L.; and Driessens, K. 2001. Relational reinforcement learning. *Machine learning* 43(1-2):7–52.

Fern, A.; Yoon, S. W.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *ICAPS*, 191–199.

Fern, A.; Yoon, S.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research* 25:75–118.

Goetschalckx, R., and Driessens, K. 2007. Cost-sensitive reinforcement learning. In *Proceedings of the workshop on AI Planning and Learning*, 1–5.

Gretton, C., and Thiébaux, S. 2004. Exploiting first-order regression in inductive policy selection. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, 217–225. AUAI Press.

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. Morgan Kaufmann Publishers Inc.

Hölldobler, S., and Skvortsova, O. 2004. A logic-based approach to dynamic programming. In *Proceedings of the Workshop on âLearning and Planning in Markov Processes–Advances and Challengesâ at the Nineteenth National Conference on Artificial Intelligence (AAAIâ04)*, 31–36.

Kersting, K., and De Raedt, L. 2003. Logical markov decision programs. In *Proc. of the IJCAIâ03 Workshop on Learning Statistical Models of Relational Data*.

Kersting, K., and Driessens, K. 2008. Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *Proceedings of the 25th international conference on Machine learning*, 456–463. ACM.

Kersting, K.; Otterlo, M. V.; and De Raedt, L. 2004. Bellman goes relational. In *Proceedings of the twenty-first international conference on Machine learning*, 59. ACM.

Levesque, H.; Pirri, F.; and Reiter, R. 1998. Foundations for the situation calculus.

Mellor, D. 2006. A learning classifier system approach to relational reinforcement learning. In *Learning Classifier Systems*. Springer. 169–188.

Moore, A. W., and Atkeson, C. G. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning* 13(1):103–130.

Morales, E. F. 2003. Scaling up reinforcement learning with a relational representation. In *Proc. of the Workshop on Adaptability in Multi-agent Systems*, 15–26.

Muller, T. J., and Van Otterlo, M. 2005. Evolutionary reinforcement learning in relational domains. In *Proceedings of the 7th European Workshop on Reinforcement Learning*.

Natarajan, S.; Khot, T.; Kersting, K.; Gutmann, B.; and Shavlik, J. 2012. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning* 86(1):25–56.

Ramon, J.; Driessens, K.; and Croonenborghs, T. 2007. Transfer learning in reinforcement learning problems through partial policy recycling. In *European Conference on Machine Learning*, 699–707. Springer.

Sanner, S., and Boutilier, C. 2012. Approximate linear programming for first-order mdps. *arXiv preprint arXiv:1207.1415*.

Sanner, S. 2005. Simultaneous learning of structure and value in relational reinforcement learning. In *Workshop on Rich Representations for Reinforcement Learning*, 57.

Sanner, S. 2006. Online feature discovery in relational reinforcement learning. In *Open Problems in Statistical Relational Learning Workshop (SRL-06)*. Citeseer.

Tesauro, G. 1995. Temporal difference learning and td-gammon. *Communications of the ACM* 38(3):58–68.

Van Otterlo, M. 2004. Reinforcement learning for relational mdps. In *In Proceedings of the Machine Learning Conference of Belgium and the Netherlands*. Citeseer.

Walker, T.; Torrey, L.; Shavlik, J.; and Maclin, R. 2007. Building relational world models for reinforcement learning. In *International Conference on Inductive Logic Programming*, 280–291. Springer.

Walker, T.; Shavlik, J.; and Maclin, R. 2004. Relational reinforcement learning via sampling the space of first-order conjunctive features. In *Proceedings of the ICML Workshop on Relational Reinforcement Learning, Banff, Canada*.

Wang, C.; Joshi, S.; and Khardon, R. 2008. First order decision diagrams for relational mdps. *Journal of Artificial Intelligence Research* 31:431–472.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

Wu, J.-H. 2007. Discovering and applying domain features in probabilistic planning. In *International Conference on Automated Planning and Scheduling, Doctoral Consortium*.