Abstract

To navigate in natural environments, a robot must decide the best action to take according to its current situation and goal. This problem usually involves uncertainty given the limitations of the sensors and actuators, and the lack of knowledge about the environment. An appropriate framework for this type of problems is based on Markov Decision Processes. Markov Decision Processes (MDPs) (Bellman, 1957) have developed as a standard method for representing uncertainty in decision-theoretic planning. Traditional MDP solution techniques have the drawback that they require an explicit state representation, limiting their applicability to real-world problems. Factored representations (Boutilier et al., 1999) address this drawback via compactly specifying the state-space in factored form as the set of
possible assignments to a set of state variables by using dynamic Bayesian networks. Such Factored MDPs allow for representing exponentially large state spaces. The algorithms for planning using MDPs, however, still run in time polynomial in the size of the state space or exponential in the number of state-variables. Recent work (Dean and Givan, 1997), (Pineau et al., 2003) use the notions of abstraction and aggregation to group states that are similar with respect to certain problem characteristics to further reduce the complexity of the solution. However, current approaches for solving MDPs still have limitations when dealing with very large or continuous state spaces. An additional problem, in particular for complex, unstructured environments, is how to define adequate abstract states and a reasonable state transition model for such representation.

In this work we propose an abstract representation based on qualitative states for continuous space MDPs and a methodology to automatically learn the model based on this representation. Based on the location of the goal(s), the state space is partitioned into a set of abstract states that represent zones of similar reward, which we denominate qualitative states. This partition is automatically obtained using decision tree learning techniques. Then, through a random exploration of the environment, a transition model is induced over the qualitative states using a factored representation and a Bayesian network learning algorithm. Thus, an abstract MDP model is learned, with a structured representation for the reward function (as a decision tree) and the transition function (as a two-stage dynamic Bayesian network), both based on qualitative states. Using this representation, we can easily solve the MDP using traditional techniques (Puterman, 1994) to obtain an optimal policy.

We have tested our method with a simulated mobile robot which has to navigate in a continuous space with regions of different rewards. Several scenarios were used, some with only one goal, other with several goals of different reward distributions, and even non-desirable zones for the robot (negative rewards). After a rapid, random exploration of the environment, for all scenarios the model was learned and solved. The policies obtained in each case are reasonable, guiding the robot to the regions of high reward and avoiding regions of low reward. We report the times for solving qualitative and discrete factored MDP versions in each case. Our approach can be applied to large, natural environments, where it is not easy to define a state abstraction and a transition model, and in which the exploration can be done using teleoperation.

The rest of the paper is organized as follows. Section 2 introduces MDPs, and section 3 related work in factored and abstract representation. In section 4, we present the qualitative MDP representation, and in section 5 the methodology to learn the model. Section 6 summarizes the experiments and results. We conclude with a summary and directions for future work.

2. Markov Decision Processes

A Markov Decision Process (MDP) (Puterman, 1994) models a sequential decision problem, in which a system evolves in time and is controlled by an agent. The system dynamics is governed by a probabilistic transition function that maps states and actions to states. At each time, an agent receives a reward that depends on the current state and the applied
action. Thus, the main problem is to find a control strategy or policy that maximizes the expected reward over time.

Formally, an MDP is a tuple $M = \langle S, A, \Phi, R \rangle$, where $S$ is a finite set of states $\{1, \ldots, n\}$. $A_s$ is a finite set of actions for each state. $\Phi : A \times S \rightarrow \Pi(S)$ is the state transition function specified as a probability distribution. The probability of reaching state $s'$ by performing action $a$ in state $s$ is written as $\Phi(a, s, s')$. $R : S \times A \rightarrow \mathbb{R}$ is the reward function. $R(s, a)$ is the reward that the agent receives if it takes action $a$ in state $s$. A policy for an MDP is a mapping $\pi : S \rightarrow A$ that selects an action for each state.

A solution to an MDP is a policy that maximizes its expected value. Two popular methods for finding an optimal policy are: (a) value iteration and (b) policy iteration (Puterman, 1994).

3. Factored and Abstract MDPs

Traditional MDP solution techniques have the drawback that they require an explicit state space, limiting their applicability to real-world problems. Factored representations address this drawback via compactly specifying the state-space in factored form. In a factored MDP, the set of states is described via a set of random variables $X = \{X_1, \ldots, X_n\}$, where each $X_i$ takes on values in some finite domain $\text{Dom}(X_i)$. The framework of dynamic Bayesian networks (DBN) (Dean and Kanasawa, 1989), (Dearden and Boutilier, 1997) gives us the tools to describe the transition function concisely. For each action, a two-stage DBN specifies the transition model. An even more compact representation can be obtained by representing the transition tables and value functions as decision trees (Boutilier et al., 1999) or algebraic decision diagrams (Hoey et al., 1999).

A further reduction in complexity can be obtained by state abstraction and aggregation techniques (Boutilier et al., 1999). Dean and Givan (1997) describe an algorithm that partitions the state space into a set of blocks such that each block is stable; that is, it preserves the same transition probabilities as the original model. Although this algorithm produces an exact partition, this could still be too complex. In many applications an approximate model could be sufficient to construct near-optimal policies. Other approaches consider problem decomposition, in which an MDP is partitioned in several problems that are solved independently and then pieced together (Boutilier et al., 1999). A similar approach is based on hierarchical clustering of states to build hierarchical MDPs (Pineau et al., 2003). An alternative extension is to allow multiple actions per decision epoch, such as in concurrent MDPs (Mausam and Weld, 2004) and multiply-sectioned MDPs (Elinas et al., 2004).

The previous approaches for solving MDPs have extended the state of the art so we can solve MDPs with thousands of states, but still have limitations when dealing with very large state spaces, and in particular for continuous domains. An additional problem, in general not considered in this area, is how to obtain the model. This has been tackled by the reinforcement learning community using experience to learn a policy and to learn a model at the same time (e.g., Dyna-Q (Sutton and Barto, 1998)). Nevertheless, model-free approaches used in reinforcement learning, such as Q-learning (Sutton and Barto, 1998)), take a very long time to converge, even with such improvements, when compared to dynamic programming techniques. Our learning approach follows a different direction. It first induces
a partition on the state space and then learns the model over such abstracted states using just a random exploration of the environment.

4. Qualitative MDPs

4.1 Qualitative states

We define a qualitative state space $Q$ as a set of states $q_1, q_2, ..., q_n$ that have different utility properties. These properties map the state space into a set of partitions, such that each partition corresponds to a group of continuous states with a similar reward value. In a qualitative MDP, a state partition $q_i$ is a region bounded by a set of constraints over the dimensions involved in the reward function. The relational operators used in this approach are $<$ and $\geq$. For example, assuming that the immediate reward is a function of the linear position in a robot navigation domain, a qualitative state could be a region in an $x$-$y$ coordinates system bounded by the constraints: $x \geq \text{val}(x_0)$ and $y \geq \text{val}(y_0)$, expressing that the current $x$ coordinate is limited by the interval $[\text{val}(x_0), \infty]$, and the $y$ coordinate by the interval $[\text{val}(y_0), \infty]$. It is evident that a qualitative state covers a large number of states with the same reward value.

Similarly to the reward function in a factored MDP, the state space $Q$ is represented by a decision tree ($Q$-tree). Each leaf in the induced decision tree is labeled with a new qualitative state. Even with leaves with the same reward value, we assign a different qualitative state value. This produces more states but at the same time creates more guidance that helps to produce more adequate policies. In particular, for robot navigation, it is also important that the states are geometrically continuous. States with similar reward are partitioned so each $q$-state is a continuous region. Figure 1 shows this trivial tree transformation in a two dimensional domain.

![Figure 1: Transformation of the reward decision tree into a Q-tree. Nodes in the tree represent continuous variables and edges evaluate whether or not this variable is greater than a particular continuous value.](image)

Each branch in the $Q$-tree denotes a set of constraints for each partition $q_i$. Figure 2 illustrates the constraints associated to the example presented above, and its representation in a 2-dimensional space.
4.2 Qualitative MDP Model Specification

We can define a qualitative MDP as a factored MDP with a set of hybrid qualitative–discrete factors. In a factored MDP, the set of states is described via a set of random variables $X = \{X_1, \ldots, X_n\}$, where each $X_i$ takes on values in some finite domain $\text{Dom}(X_i)$. A state $x$ defines a value $x_i \in \text{Dom}(X_i)$ for each variable $X_i$. Even though, it is still possible that the set of states $S = \text{Dom}(X_i)$ is exponentially large, making it impractical to represent the transition model explicitly as matrices. Fortunately, the framework of dynamic Bayesian networks (DBN) (Dean and Kanasawa, 1989; Darwiche and M., 1994) gives us the tools to describe the transition function concisely. This representation includes the conditional probability of each post-action node (at time $t+1$) given its parents’, under the effects of an action. There is one DBN for each action. In figure 3 we show all the actions in the same network (to save space).

In the context of factored representations, the qualitative state space $Q$, described in section 4.1, is a factor that concentrates all the continuous variables involved in the reward function. The idea is to substitute all these variables by this abstraction to reduce the dimensionality of the state space. The role of factor $Q$ under the influence of the reward function in the transition model is also shown in figure 3.

We next describe how this qualitative MDP model is learned.

5. Learning Qualitative MDPs

In practice, it is difficult to specify a qualitative MDP model for a particular problem. However, there are cases where this specification is extracted more naturally from data using machine learning algorithms. In this work, we approximate the domain reward function using J48, a Java re-implementation of C4.5 (Quinlan, 1993) included in Weka (Witten, 2005). To start this process, examples must be stated such that the domain variables are the attributes, and the immediate reward value is the class. The classification is then made using a representative set of samples from the system behavior.
Figure 3: A simple DBN representing a transition function with a $Q$ variable and 3 discrete variables, $d_i$. $R$ represents the reward function that is used to generate the q-states in $Q$.

In order to induce the factored transition model, we used the Java implementation of the K2 algorithm for learning Bayesian networks (Cooper and Herskovits, 1992) that is included in Elvira (Consortium, 2002). A 2-stage dynamic Bayesian network representing the transition model is trained with data samples from the system dynamics. State transitions can be obtained either by an operator acting on the system, or by an automatic control system. In this case, the set of attributes are the domain variables at time $t$, the domain variables at time $t+1$, and the action that produced this temporal change.

In summary, given a set of state transition represented as a set of random variables, $O_j = \{X_{j1}, \ldots, X_{jn}\}$, for $j = 1, 2, \ldots, M$, for each state, an action $A$ executed by an operator, and a reward (or cost) $R^j$ associated to each transition, we proposed a methodology for learning a qualitative factored MDP model:

1. From a set of examples $\{O, R\}$ obtain a decision tree, $RDT$, that predicts the reward function $R$ in terms of continuous state variables, $X_1, \ldots, X_n$.

2. Obtain from the decision tree, $RDT$, the set of constraints relevant to determine the qualitative states (q-states) in the form of a Q-tree. In terms of the domain variables, we obtain a new variable $Q$ representing the reward-based qualitative state space whose values are the q-states.

3. Discretize the remaining continuous domain variables, $d_i$ (those not included in the reward function), according to a domain knowledge criteria. The original discrete domain variables remain the same.

4. Qualify data from the original sample in such a way that the new set of attributes are the Q variable, the remaining discretized state variables not included in the decision tree, and the action $A$. This transformed data set can be called the qualified data set.
5. Format the qualified data set in such a way that the attributes follow a temporal causal ordering. For example variable $Q_t$ must be set before than $Q_{t+1}$, $X_1_t$ before than $X_1_{t+1}$, and so on. The whole set of attributes should be the variable Q in time t, the remaining system variables in time t, the variable Q in time t+1, the remaining system variables in time t+1, and the action $A$.

6. Prepare data for the induction of a 2-stage dynamic Bayesian net. According to the action space dimension, split the qualified data set into $|A|$ sets of samples for each action.

7. Induce the transition model for each action from $A$ using the K2 algorithm.

Once the model is learned, it can be solved using value iteration to obtain the optimal policy.

6. **Experimental Results**

We tested our approach in a robot navigation domain using a simulator. The robot sensor system included x-y position, and navigation bounds detection. The possible actions are discrete orthogonal movements of 50 cm to the right, left, up, and down. Figure 4 shows an example of a 10 m$^2$ navigation area. In this example, goals are represented as different blue-scale regions where darker blue colors (dark grey) represent higher positive rewards. Similarly, non-desirable regions are illustrated by yellow colors, where darker yellow color represent more negative rewards (light grey). The remaining regions in the navigation area receive 0 reward (black). Since obstacles are not considered robot states, they are not included. The planning problem is to automatically obtain an optimal policy for the robot to achieve its goals.

![Figure 4: Robot navigation area showing the distribution of goals and non-desirable zones.](image)

The simulated free-flying robot is located at the left-bottom corner.

Figure 5, shows the induced decision tree for the reward function. Leaves are the qualitative states with different rewards, and branches are geometrical constraints for each qualitative partition. Figure 6 (left) illustrates the state space partition resulting from grouping regions with similar rewards. From the qualitative states, a transition model in the form of a DBN was induced using K-2. The qualitative MDP solution described of this factored MDP is showed in figure 6 (right).
Figure 5: Induced decision tree that approximates the reward function.

Figure 6: State space partition resulting from grouping regions with similar rewards (left). Optimal policy obtained by the value iteration algorithm (right).
We compared our approach against factored MDPs on different problems. Table 1 shows the results of four instances of a 10 m² domain with a uniform partition of 50 cm² and the partition induced with decision trees. As can be seen, substantial time reductions can be achieved by our approach under different conditions.

<table>
<thead>
<tr>
<th>Rewards Regions</th>
<th>Factored MDP</th>
<th>Factored Qualitative MDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>Region size (cm)</td>
<td>Rewards Cells</td>
</tr>
<tr>
<td>0, 0, 200</td>
<td>200</td>
<td>2</td>
</tr>
<tr>
<td>0, 200, 200, 400</td>
<td>200</td>
<td>4</td>
</tr>
<tr>
<td>0, 200, 200</td>
<td>200</td>
<td>26</td>
</tr>
<tr>
<td>0, 400, 400, 400</td>
<td>200</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 1: Comparison of our approach versus a discrete factored MDP under different problem configurations.

7. Conclusions

The main contributions of this paper are an abstract representation based on qualitative states for continuous space MDPs and a methodology to automatically learn the model based on this representation. Based on a reward structure the state space is partitioned into a set of abstract states, which we denominate qualitative states. This partition is automatically obtained using decision tree learning techniques. Then, through a random exploration of the environment, a transition model is induced over the qualitative states using a factored representation and a Bayesian network learning algorithm. Thus, an abstract MDP model is learned, with a structured representation for the reward and transition functions. Using this representation we solve the MDP using standard techniques.

We have tested this methodology using a mobile robot simulator for several scenarios. In every case, we obtained a quasi-optimal policy, in approximately two orders of magnitude less time than using a discrete, factored representation. The solution is obtained automatically just based on the position of the reward regions and a random exploration of the environment. Thus, we consider that this approach can be applied to solve navigation problems in complex, unstructured environments.

In the future, we plan to improve the partition of the state space by refining the partitions based on the transition or utility functions.

References


