Machine Learning Techniques: A Tool for Learning and Planning

Akinbode K.B
Computer science department
The Polytechnic Ibadan, Oyo state, Nigeria.
akinbodk@hotmail.com

ABSTRACT

To invest in natural environments, an investor must decide the best action to take according to its current situation and goal, a problem that can be represented as a Markov Decision Process (MDP). In general, it is assumed that a reasonable state representation and transition model can be provided by the user to the system.

When dealing with complex domains, however, it is not always easy or possible to provide such information. In this paper, a system is described that can automatically produce a state abstraction and can learn a transition function over such abstracted states, called qualitative states.

A qualitative state is a group of states with similar properties and rewards. They are induced from the reward function using decision trees. The transition model represented as a MDP is learned using a Bayesian network learning algorithm.

The outcome of this combined learning process produces a very compact MDP that can be efficiently solved using standard techniques. We show experimentally that this approach can learn efficiently a reasonable policy that an investor can invest upon in large and complex domains.

KEYWORDS: Markov decision process, transition model, qualitative states, decision trees and Bayesian network learning algorithm.

1. INTRODUCTION

To invest in natural environments, an investor must decide the best action to take according to its current situation and goal. An appropriate framework for this type of problems is based on Markov Decision Processes.

Markov Decision Processes (MDPs) [1] "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 [3] 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, [3] "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 goal(s) of an investor, 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 [4] to obtain an optimal policy.

We have tested our method with a simulated mobile agent which has to invest in a continuous state space with rewards for each action. Several scenarios were used, some with only one goal,

other with several goals of different reward distributions, and even non-desirable zones for the investors (negative rewards). After a rapid, random exploration of the environment, for all scenarios the model was learned and tested. The policies obtained in each case are reasonable, guiding the investor 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.

The rest of the paper is organized as follows. Section 2 introduces MDPs, and section 3 related works 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.

2. MARKOV DECISION PROCESSES

A Markov Decision Process (MDP) [2] "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

$M = [S, A_s, \phi, R]$ (i)

where:

- S is a finite set of states {1, . . . , n}
- A_s is a finite set of actions for each state.
- ϕ 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 ϕ (a, s, s′).
- 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 MDP is a mapping that selects an action for each state. A solution to 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 [14].

3. RELATED WORKS

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 = {X1, .., Xn}, where each Xi takes on values in some finite domain Dom(Xi). The framework of dynamic Bayesian networks (DBN) [5], [6] 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 [7] or algebraic decision diagrams [8].

A further reduction in complexity can be obtained by state abstraction and aggregation techniques [9]. 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 [10]. A similar approach is based on hierarchical clustering of states to build hierarchical MDPs [11].

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 [12]). Nevertheless, model-free approaches used in reinforcement learning, such as Q-learning [13], 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 qi 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 ≥. For example, a qualitative state could be a region in an x–y coordinates system bounded by the constraints:

$x \geq val(x0)$ and $y \geq val(y0)$, expressing that the current x coordinate is limited by the interval $[val(x0),\infty]$, and the y coordinate by the interval $[val(y0),\infty]$. 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 leave in the induced decision tree is labelled 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 investment problem, it is also important that the states are continuous. States with similar reward are partitioned so each q–state is a continuous region.

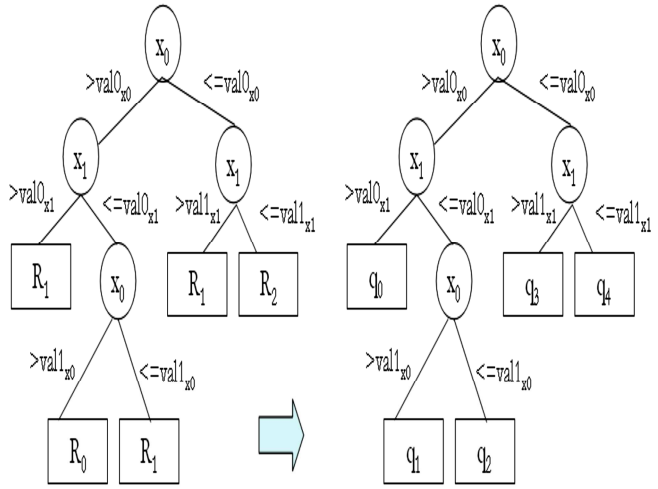

Figure 1: Transformation of the reward decision tree into a Q-tree

Nodes in the tree represent continuous variables and edges evaluate whether or not this variable is greater than a particular continuous value.

Each branch in the Q–tree denotes a set of constraints for each partition qi. Figure 2 illustrates the constraints associated to the example presented above, and its representation in a 2-dimensional state space.

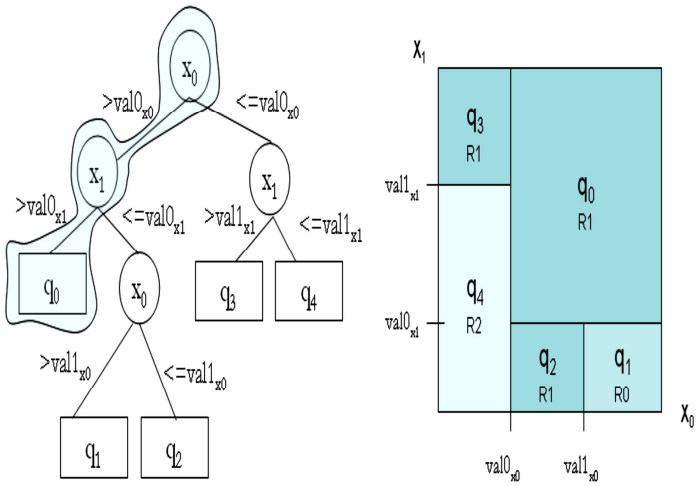

Figure 2: In a Q-tree, branches are constraints and leaves are qualitative states. A graphical representation of the tree is also shown. Note that when an upper or lower variable bound is infinite, it must be understood as the upper or lower variable bound in the domain.

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 $X = \{X1, \ldots, Xn\}$, where each Xi takes on values in some finite domain Dom(Xi). A state x defines a value $xi \in Dom(Xi)$ for each variable Xi. Even though, it is still possible that the set of states $S = Dom(Xi)$ is exponentially large, making it impractical to represent the transition model explicitly as matrices. Fortunately, the framework of dynamic Bayesian networks (DBN) [17] 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.

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 [18] included in Weka [19]. 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 behaviour.

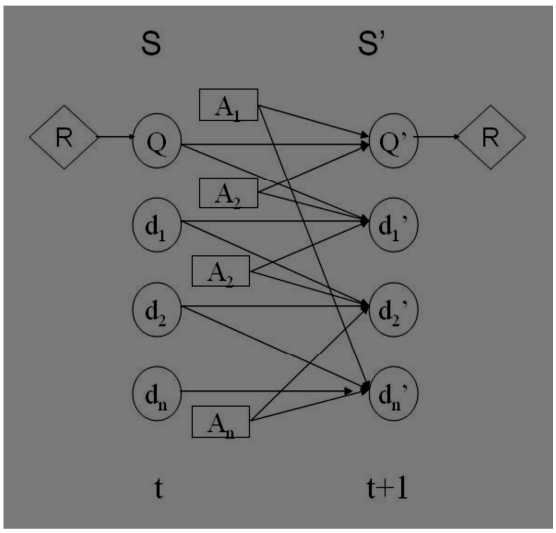

Figure 3: A simple DBN representing a transition function with a Q variable and 3 discrete variables, di. R represents the reward function that is used to generate the q -states in Q.

In order to induce the factored transition model, we used the Java implementation of the K2 algorithm for learning Bayesian network [15] that is included in Elvira [16]. 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, $O^j = \{X^j_1, .., X^j_n\}$, for j = 1, 2, ...,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:

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

- 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.
- 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}, $X1_t$ before than $X1_{t+1}$, 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.
- 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.
- Induce the transition model for each action from A using the K2 algorithm.

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

6. EXPERIMENTAL RESULTS

We tested our approach in an investment problem domain using a simulator. The investment system included x-y position, and navigation bounds detection. The possible actions are purchasing shares and selling shares. Figure 4 shows how goals or profits are represented as different blue-scale regions where darker blue colours (dark grey) represent higher positive rewards. Similarly, non-desirable regions or loss are illustrated by yellow colours, where darker yellow colours represent more negative rewards (light grey). The remaining regions in the navigation area receive 0 reward (black). Since obstacles or sentiments are not considered, they are not included. The planning problem is to automatically obtain an optimal policy for an investor to achieve its goals.

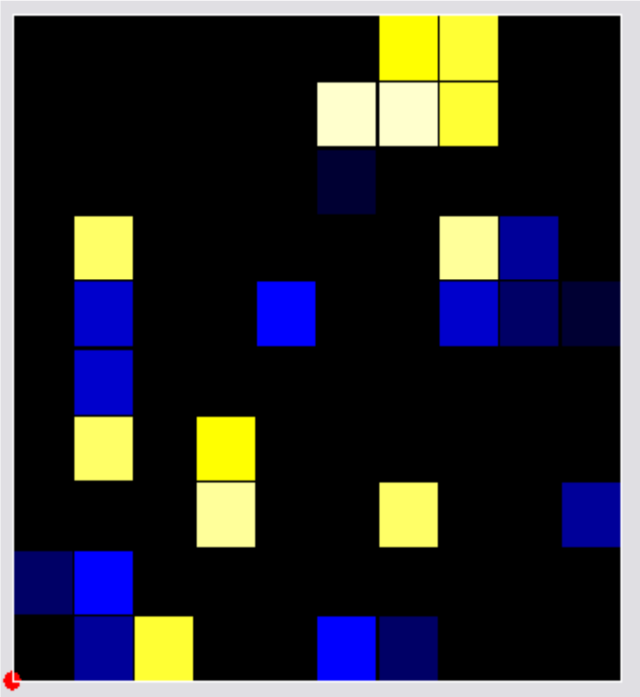

Figure 4: Investment navigation area showing the distribution of goals and non-desirable zones. The simulated investor is located at the left-bottom corner.

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).

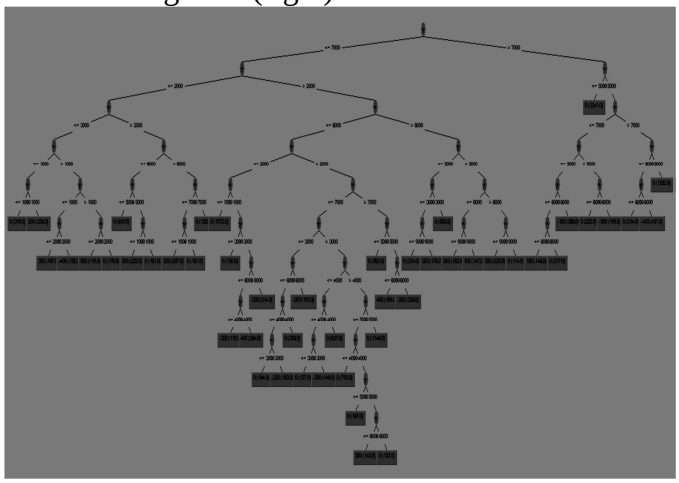

Figure 5: Induced decision tree that approximates the reward function.

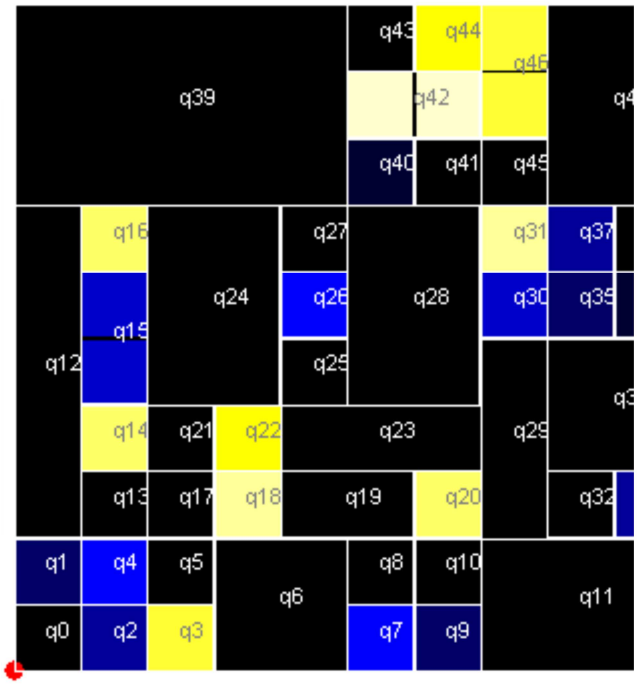

Figure 6: State space partition resulting from grouping regions with similar rewards.

7. CONCLUSION

The main contributions of this paper is to propose an abstract representation for investment problem 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.

This methodology is tested using a mobile agent 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 investment problems in complex and 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.

8. REFERENCES

1. R.E. Bellman Dynamic Programming. Princeton U. Press, Princeton, N.J., 1957.
2. C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: structural assumptions and computational leverage. Journal of AI research, 11:1-94, 1999
3. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: an anytime algorithm for POMDPs. In International Joint Conference on Artificial Intelligence (IJCAI), pages 1025–1032, 2003.
4. Elvira Consortium. Elvira: an environment for creating and using probabilistic graphical models. Technical report, U. de Granada, Spain, 2002
5. G. F. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. Machine learning, 1992.
6. P. S. Castro and D. Precup. Using linear programming for bayesian exploration in markov decision processes. In International Joint Conference on Artificial Intelligence (IJCAI), pages 2437–2442, 2007.
7. M. J. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In International Joint Conference on Artificial Intelligence (IJCAI), pages 1324–1331, 1999.
8. R. Dearden, N. Friedman, and S. J. Russell. Bayesian Q-learning. In AAAI Conference on Artificial Intelligence, pages 761–768, 1998.
9. R. Dearden, N. Friedman, and D. Andre. Model based bayesian

exploration. In Conference on Uncertainty in Artificial Intelligence (UAI), pages 150–159, 1999.

10. E. Delage and S. Mannor. Percentile optimization in uncertain mdps with application to efficient exploration. In International Conference on Machine Learning (ICML), 2007.
11. F. Doshi, J. Pineau, and N. Roy. Reinforcement learning with limited reinforcement: Using Bayes risk for active learning in POMDPs. In International Conference on Machine Learning, pages 256–263. ACM, 2008.
12. F. Doshi-Velez. The infinite partially observable markov decision process. In Neural Information Processing Systems (NIPS), volume 22, 2010.
13. A. Doucet, N. de Freitas, and N. Gordon. Sequential Monte Carlo Methods In Practice. Springer, 2001.
14. M. Duff. Monte-Carlo algorithms for the improvement of finite-state stochastic controllers: Ap- plication to bayes-adaptive Markov decision processes. In International Workshop on Artificial Intelligence and Statistics (AISTATS), 2001.
15. M. Duff. Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Pro- cesses. PhD thesis, University of Massachusetts Amherst, Amherst, MA, 2002.
16. Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The gaussian process approach to temporal difference learning. In International Conference on Machine Learning (ICML), pages 154–161, 2003.
17. Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with gaussian processes. In International Conference on Machine learning (ICML), pages 201–208, 2005.
18. A. A. Feldbaum. Dual control theory, parts i and ii. Automation and Remote Control, 21:874–880 and 1033–1039, 1961.
19. O. Zane. Discrete-time bayesian adaptive control problems with complete information. In IEEE Conference on Decision and Control, pages 2748–2749, 1992.
20. T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans. Bayesian sparse sampling for on-line reward optimization. In International Conference on Machine learning (ICML), pages 956–963, 2005.