Active Learning of Dynamic Bayesian Networks in Markov Decision Processes

Anders Jonsson[1] and Andrew Barto[2]

[1] Department of Information and Communication Technologies
Universitat Pompeu Fabra
Passeig de Circumval·lació, 8
08003 Barcelona, Spain
anders.jonsson@upf.edu
[2] Autonomous Learning Laboratory
Department of Computer Science
University of Massachusetts
Amherst MA 01003, USA
barto@cs.umass.edu

Abstract. Several recent techniques for solving Markov decision processes use dynamic Bayesian networks to compactly represent tasks. The dynamic Bayesian network representation may not be given, in which case it is necessary to learn it if one wants to apply these techniques. We develop an algorithm for learning dynamic Bayesian network representations of Markov decision processes using data collected through exploration in the environment. To accelerate data collection we develop a novel scheme for active learning of the networks. We assume that it is not possible to sample the process in arbitrary states, only along trajectories, which prevents us from applying existing active learning techniques. Our active learning scheme selects actions that maximize the total entropy of distributions used to evaluate potential refinements of the networks.

1 Introduction

Existing solution techniques for Markov decision processes, or MDPs, scale poorly to tasks with large state spaces. A major research challenge is to develop techniques that exploit the structure of a task and reduce the size of the state space. A common type of structure is factored state, which means that the available information belongs to distinct categories. For example, a robot navigating through a building can usually distinguish between its location, the object it is holding, and the energy level of its battery, instead of perceiving the current situation as a single observation. Factored MDPs use a set of state variables to represent the state in a way that is more appropriate for tasks of this type. Dynamic Bayesian networks, or DBNs [1], are particularly well suited for exploiting structure in factored MDPs by capturing conditional independence between state variables as a result of executing actions. Several researchers have developed algorithms for solving factored MDPs that exploit structure expressed by DBNs [2–6].

It is unrealistic to assume that a DBN model is always available prior to solving an MDP. We address the non-trivial problem of learning DBNs from experience. There

exist algorithms in the literature for learning the structure of Bayesian networks [7–9]. However, these algorithms assume that a data set is given, whereas solution techniques for MDPs typically have to gather data in the form of transitions and reward through interaction with the environment. The complexity of learning DBNs depends heavily on the time it takes to collect data. It is possible to accelerate data collection by selecting high-quality data instances through a process called active learning. There exist several techniques for active learning of Bayesian networks [10–12]. These techniques perform experiments by clamping a subset of the variables to fixed values and sampling over the remaining variables.

A robot exploring its environment for the first time cannot transport itself to any location instantaneously. Instead, it must wander around to try the effect of different actions in different places. In this work, we assume that it is only possible to sample MDPs along trajectories, not in arbitrary states. In other words, the only way to gather information about transitions and reward is by repeatedly executing an action in the current state. Since it is not possible to simulate the effect of actions in hypothetical states, we cannot perform experiments by clamping a subset of the variables to fixed values. Consequently, we cannot apply existing techniques for active learning. However, there is still an opportunity to perform active learning of DBNs in factored MDPs. Because the DBN model of a factored MDP consists of one DBN for each action, by selecting an action we effectively select a DBN to collect data for. As a consequence, we can consider policies for action selection whose aim is to gather data as quickly and efficiently as possible. As far as we know, there exists no previous work for learning DBN models of factored MDPs under these assumptions.

1.1 Overview of our work

We use trees to represent the conditional probabilities of the DBNs, and develop an algorithm that implicitly learns the DBNs by growing the conditional probability trees. Our algorithm collects data instances by executing actions and grows the trees as soon as a minimum number of data instances correspond to each relevant value of each split variable. The minimum number is defined by a threshold parameter, and potential refinements are evaluated as soon as the threshold is exceeded. The algorithm uses the Bayesian Information Criterion (BIC) [13] and the likelihood-equivalent Bayesian Dirichlet metric (BDe) [9] to evaluate potential refinements. We assume that no data is available to begin with and develop a technique for active learning of DBNs to accelerate data collection. The time to collect data is minimized if the distribution of data instances across values of each potential split variable is perfectly uniform. We use the entropy of the distributions to measure uniformity and select actions that maximize the total entropy of the distributions.

In some tasks, the BIC and BDe scores fail to detect most of the refinements necessary to learn an accurate DBN model. This typically happens when the effect of actions depends on many state variables. Since the BIC and BDe scores penalize trees with many leaves, the algorithm prefers to keep the size of the trees small instead of continuing to refine the trees. This is a serious issue since algorithms that take advantage of DBNs to solve factored MDPs depend on an accurate DBN model. We address this issue by applying regularization [14] to the BIC score. The BIC score is composed of a

log likelihood term and a penalty term. This quantity fits nicely into the regularization framework if we multiply the penalty term by a parameter λ. Results show that varying λ can increase the accuracy of the learned DBN model.

Our work is related to the problem of exploration in reinforcement learning [15]. Existing exploration techniques do not learn DBN models of MDPs. Since there exist several efficient algorithms that use DBNs to solve factored MDPs, there is a benefit to learning this representation. Ours is an undirected approach that does not require enumeration of the state space, as opposed to directed exploration, which maintains relevant information for each state. Since we want to scale to large state spaces, we do not want to store quantities whose size is proportional to the number of states.

2 Bayesian networks

Let $\mathbf{X}$ be a set of discrete variables, and let $\mathbf{x}$ be an assignment of values to the variables in $\mathbf{X}$. Let $f_{\mathbf{Y}}$, $\mathbf{Y} \subseteq \mathbf{X}$, be a projection such that if $\mathbf{x}$ is an assignment to $\mathbf{X}$, $f_{\mathbf{Y}}(\mathbf{x})$ is $\mathbf{x}$'s assignment to $\mathbf{Y}$. A Bayesian network (BN) $B = \langle G, \theta \rangle$ consists of a directed acyclic graph G with one node per variable $X_i \in \mathbf{X}$ and a set of parameters θ defining the conditional probabilities of the variables. The joint probability distribution of the variables is given by:

$$P(\mathbf{x}) = \prod_i P(X_i = f_{\{X_i\}}(\mathbf{x}) \mid \mathbf{Pa}(X_i) = f_{\mathbf{Pa}(X_i)}(\mathbf{x})),$$

where $\mathbf{Pa}(X_i) \subset \mathbf{X}$ is the subset of parent variables of X_i, i.e., variables with edges to X_i in G, and the probabilities $P(X_i = f_{\{X_i\}}(\mathbf{x}) \mid \mathbf{Pa}(X_i) = f_{\mathbf{Pa}(X_i)}(\mathbf{x}))$ are defined by parameters in θ.

A dynamic Bayesian network, or DBN [1], is a Bayesian network that models the evolution of a set of variables in a temporal process. The directed acyclic graph of a DBN has two layers of nodes: one layer representing the current values of the variables, and one layer representing the next values of the variables. The edges between layers are unidirectional and always point from the current layer to the next layer. There can also be edges between nodes within a layer.

Structure learning is the problem of finding the BN that best fits a data set $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. A common approach is to compute the posterior probability distribution $P(B \mid D)$ over BNs and choose the BN that maximizes $P(B \mid D)$. Two common approximations of $P(B \mid D)$ are the Bayesian Information Criterion (BIC) [13] and the likelihood-equivalent Bayesian Dirichlet metric (BDe) [9]. From Bayes theorem it follows that $P(B \mid D) \propto P(D \mid B)P(B)$. The BIC score makes the approximation

$$\log[P(D \mid B)P(B)] \approx L(D \mid B) - \frac{|\theta|}{2} \log |D|, \tag{1}$$

where $L(D \mid B)$ is the log likelihood of D given B. If the data set D contains no missing values, the log likelihood decomposes as

$$L(D \mid B) = \sum_i \sum_j \sum_k N_{ijk} \log \theta_{ijk},$$

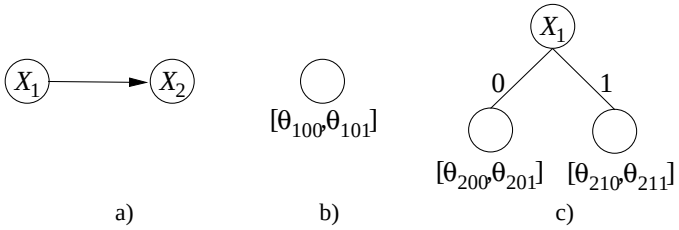

Fig. 1. a) Graph G of a BN with two variables; conditional probability trees for b) X_1, c) X_2

where N_{ijk} is the number of data points $\mathbf{x} \in D$ such that $f_{\mathbf{Pa}(X_i)}(\mathbf{x}) = j$ and $f_{\{X_i\}}(\mathbf{x}) = k$, and $\theta_{ijk} = P(X_i = k \mid \mathbf{Pa}(X_i) = j)$. The log likelihood is maximized for $\theta_{ijk} = N_{ijk} / \sum_k N_{ijk}$. The BDe score makes the approximation

$$P(D \mid B)P(B) \approx \prod_i \prod_j \frac{\Gamma(\sum_k N'_{ijk})}{\Gamma(\sum_k [N'_{ijk} + N_{ijk}])} \prod_k \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})}, \qquad (2)$$

where N'_{ijk} are hyperparameters of a Dirichlet prior and $\Gamma(x)$ is the Gamma function.

Finding the BN with highest BIC or BDe score is NP-complete [16]. However, both scores decompose into a sum of terms for each variable X_i and each value j and k (we need to take the logarithm of BDe first). The score only changes locally when we add or remove edges between variables in G. Researchers have developed hill-climbing algorithms that perform greedy search to find high-scoring BNs by repeatedly adding or removing edges between variables in G [7, 9]. These algorithms have been extended to DBNs [8].

As an example, consider a BN with two binary variables X_1 and X_2. Assume that we have collected three data points $(0, 0)$, $(0, 1)$, and $(1, 1)$. Also assume that the BN has an edge from X_1 to X_2, and that we use trees to store the conditional probabilities of X_1 and X_2. Figure 1 shows the graph G of the BN as well as the conditional probability trees for X_1 and X_2. Since X_1 has no parents in G, the count N_{100} simply indicates the number of data points that assign 0 to X_1. In this case, $N_{100} = 2$ and $N_{101} = 1$. The log likelihood is maximized for $\theta_{100} = N_{100}/(N_{100} + N_{101}) = 2/3$ and $\theta_{101} = 1/3$. One data point, $(0, 0)$, assigns the value 0 to X_1 and 0 to X_2, so $N_{200} = 1$. Likewise, $N_{201} = 1$, $N_{210} = 0$, and $N_{211} = 1$. The log likelihood is maximized for $\theta_{200} = N_{200}/(N_{200} + N_{201}) = 1/2$, $\theta_{201} = 1/2$, $\theta_{210} = 0$, and $\theta_{211} = 1$.

The BIC score for the BN is given by the expression

$$\sum_i \sum_j \sum_k N_{ijk} \log \theta_{ijk} - \frac{|\theta|}{2} \log |D| =$$
$$= 2 \log \frac{2}{3} + 1 \log \frac{1}{3} + 1 \log \frac{1}{2} + 1 \log \frac{1}{2} + 0 + 1 \log 1 - \frac{6}{2} \log 3.$$

Note that each leaf of the conditional probability tree for variable X_i contributes to the BIC score with a term $\sum_k N_{ijk} \log \theta_{ijk} - \frac{|Dom(X_i)|}{2} \log |D|$, where $Dom(X_i)$ is the

domain of X_i and j is the assignment of values to the parents of X_i in G as indicated by the path from the root to the leaf. Also note that the contribution from each leaf is smaller than 0, and that it is maximized when all data points assign the same value to X_i, in which case the first term equals 0. For example, the contribution from the right leaf in the conditional probability tree for X_2 is $0 + 1 \log 1 - \frac{2}{2} \log 3 = 0 - \log 3$. The intuition is that the higher the BIC score, the more accurately we can predict the value of X_i given the values of its parents in G.

3 Markov decision processes

A finite Markov decision process (MDP) is a tuple $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$, where S is a finite set of states, A is a finite set of actions, $\Psi \subseteq S \times A$ is a set of admissible state-action pairs, P is a transition probability function, and R is an expected reward function. As a result of executing action $a \in A_s \equiv \{a' \in A \mid (s, a') \in \Psi\}$ in state $s \in S$, the process transitions to state $s' \in S$ with probability $P(s' \mid s, a)$ and receives an expected reward $R(s, a)$. In the discounted case, a solution to an MDP is a stochastic policy π that, for each $t > 0$, maximizes the expected return $R_t = E\{\sum_{k=t}^{\infty} \gamma^{k-t} R(s^k, a^k)\}$, where $\gamma \in (0, 1]$ is a discount factor, by selecting action a^k with probability $\pi(s^k, a^k)$ in each state s^k.

A factored MDP is described by a set of state variables $\mathbf{S}$. We use the coffee task [2], in which a robot has to deliver coffee to its user, as an example of a factored MDP. The coffee task is described by six binary state variables: $S_{\mathtt{L}}$, the robot's location (office or coffee shop); $S_{\mathtt{U}}$, whether the robot has an umbrella; $S_{\mathtt{R}}$, whether it is raining; $S_{\mathtt{W}}$, whether the robot is wet; $S_{\mathtt{C}}$, whether the robot has coffee; and $S_{\mathtt{H}}$, whether the user has coffee. Let $\{i, \overline{i}\}$ be the values of state variable S_i, where $\overline{L}$ is the office and L the coffee shop. An example state is $\mathbf{s} = (\overline{L}, U, R, \overline{W}, C, \overline{H})$. The robot has four actions: `GO`, causing its location to change and the robot to get wet if it is raining and it does not have an umbrella; `BC` (buy coffee) causing it to hold coffee if it is in the coffee shop; `GU` (get umbrella) causing it to hold an umbrella if it is in the office; and `DC` (deliver coffee) causing the user to hold coffee if the robot has coffee and is in the office. All actions have a chance of failing. The robot gets a reward of 0.9 when the user has coffee plus a reward of 0.1 when it is dry.

3.1 DBN model of factored MDPs

The DBN model of a factored MDP [2] contains one DBN for each action $a \in A$. Like the original model, we assume that the conditional probabilities of the DBNs are represented using trees as opposed to tables. This allows for a more compact representation of the conditional probabilities. Figure 2 shows the DBN for action `GO` in the coffee task. Assuming action `GO` is executed at time t, the DBN determines the resulting values of state variables at time $t + 1$. For each state variable S_i, there are two nodes in the DBN: one node S_i^t representing the value of S_i at time t, and one node S_i^{t+1} representing its value at time $t + 1$. The same is true for the expected reward R. The value of S_i at time $t + 1$ depends on the values of state variables that have edges to S_i^{t+1} in the DBN. A dashed line indicates that a state variable is unaffected by `GO`.

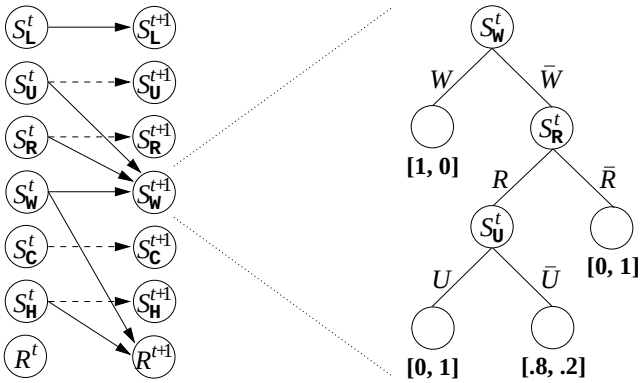

Fig. 2. The DBN for action GO in the coffee task

Figure 2 also illustrates the conditional probability tree associated with state variable $S_{\mathtt{W}}$ and action GO, which we denote $\mathcal{T}_{\mathtt{W}}^{\mathtt{GO}}$. At each leaf, the first value represents the probability that the robot is wet after executing GO, while the second value represents the probability that the robot is dry. At time t, if the robot is not wet ($\overline{W}$), it is raining (R), and the robot does not have an umbrella ($\overline{U}$), the conditional probability tree indicates that the robot is wet at time $t+1$ with probability 0.8. We assume that there are no edges between state variables at a same time step. The transition probabilities are given by $P(\mathbf{s}^{t+1} \mid \mathbf{s}^t, a) = \prod_i P_a(S_i^{t+1} = f_{\{S_i^{t+1}\}}(\mathbf{s}^{t+1}) \mid \mathbf{Pa}(S_i^{t+1}) = f_{\mathbf{Pa}(S_i^{t+1})}(\mathbf{s}^t))$, where P_a is the joint probability distribution represented by the DBN for action a.

4 Learning a DBN model

We develop an algorithm for learning DBN models of factored MDPs through interaction with the environment. Our algorithm builds a tree $\mathcal{T}_i^a$ for each pair of a state variable S_i and action a, approximating the conditional probabilities of S_i^{t+1} as a result of executing a. The family of trees for a implicitly defines the DBN for a. There is an edge between state variables S_j^t and S_i^{t+1} in the DBN if at least one node in $\mathcal{T}_i^a$ distinguishes between values of S_j^t. To build the tree $\mathcal{T}_i^a$, the algorithm starts with a small tree and collects data by executing actions in the environment. Each time action a is executed, the algorithm records a data instance consisting of the former state, the resulting state, and the reward received. Each data instance maps to exactly one leaf of $\mathcal{T}_i^a$, at which it is stored. We say that a leaf is empty if its corresponding set of data instances is empty.

A refinement at a leaf distinguishes between values of a state variable S_j^t and introduces a new leaf of $\mathcal{T}_i^a$ for each value of S_j^t. S_j^t is only considered for refinement if no internal nodes on the path from the root to the leaf of $\mathcal{T}_i^a$ already distinguish between values of S_j^t. As we have already mentioned, the BIC and BDe scores decompose into a local score for each leaf. Our algorithm evaluates a refinement by comparing the total

score of the new leaves with the score of the old leaf. If at least one refinement increases the overall score, the algorithm retains the refinement that results in the largest increase. Regardless of the outcome, data instances at the old leaf are discarded. Our approach is more sophisticated than adding edges in the graph of the DBN, since trees store conditional probabilities more compactly than tables.

Evaluating a refinement using the BIC and BDe scores really amounts to performing a statistical test to compare the posterior probabilities of two Bayesian networks given the data. It is well known that the accuracy of statistical tests, such as Chi-square, depends on having enough examples in each bin. At each leaf, and for each potential split variable S_j^t, the algorithm maintains a distribution vector M. Each entry M_k of the vector indicates the number of data instances at the leaf that assign the value k to S_j^t. When the algorithm evaluates a refinement over S_j^t, the distribution vector M determines how the data instances at the leaf will be distributed to the new leaves of $\mathcal{T}_i^a$. We define a threshold parameter K and let our algorithm evaluate a refinement as soon as at least K data instances map to each non-empty leaf for each split variable.

In some tasks, the BIC and BDe scores fail to detect most of the refinements necessary to learn an accurate DBN model. The BIC score in Equation (1) is composed of a log likelihood term, which measures the likelihood of the data given a network, and a penalty term, which penalizes a network for having many parameters. The penalty term causes the BIC score to be less sensitive to improvements to the log likelihood since each refinement increases the number of parameters. We use regularization [14] to address this issue. In regularization, a functional is defined as the sum of a fidelity term and a stabilizer term. The stabilizer term is weighted by a parameter λ. We can multiply the penalty term of the BIC score by a parameter λ to put it in the form of a fidelity term and a stabilizer term:

$$\log[P(D \mid B)P(B)] \approx L(D \mid B) - \lambda \frac{|\theta|}{2} \log |D|, \tag{3}$$

such that λ controls the magnitude of the penalty for having many parameters.

4.1 Active learning

Efficient data collection should gather sufficient data as quickly as possible. Since our algorithm requires at least K data instances to map to each non-empty leaf, the distribution of data instances across potential new leaves should be as uniform as possible for each possible refinement. The more skewed the distribution, the longer it takes to collect sufficient data to evaluate refinements. We devise the following scheme for active learning of DBNs. Before executing action a, the current state determines which leaf of $\mathcal{T}_i^a$ the resulting data instance will map to. When deciding which action to execute, we look at how the distribution vectors at corresponding leaves would change as a result of executing each action. To evaluate the change, we compute the entropy $H(M)$ of each distribution vector M:

$$H(M) = -\sum_k \theta_k \log \theta_k,$$

where $\theta_k = M_k / \sum_j M_j$. $H(M)$ is a non-negative function which is maximized when all entries of M are equal. An increase in $H(M)$ means that the distribution is becoming

more uniform; a decrease means that it is becoming more skewed. The change in $H(M)$ can be computed in constant time. In each state, the active learning scheme selects the action with largest total increase in the value of $H(M)$. With probability $\epsilon \in [0, 1]$, or if no action results in an increase of $H(M)$, the scheme selects a random action.

By maximizing the entropy, our active learning scheme maintains uniform distributions at the leaves, which in turn causes evaluation to occur as quickly as possible. However, the time it takes to collect data also depends on how often leaves are visited. The proposed scheme only implicitly affects the frequency with which leaves are visited, and assumes that each leaf is visited relatively frequently. Our approach is motivated by the fact that we want to use local information only to guide exploration. We believe that under this constraint, the entropy measure is best suited for the problem. By storing global information about the frequency with which leaves have been visited, it would be possible to try to steer exploration towards the least visited areas of the state space, although it is unclear how the system would know how to get to these areas.

5 Results

We ran experiments with our DBN learning approach in the coffee task [2], the Taxi task [17], and a simplified autonomous guided vehicle (AGV) task [18]. In each task, we compared our active learning scheme with passive learning, i.e., random action selection. In both cases, we used our approach for growing the conditional probability trees to implicitly learn the DBNs. Note that because of our assumption regarding data collection there is no meaningful way to compare our results to existing techniques for active learning. We had access to the true DBN model of each task and knew how many refinements of the trees were necessary to learn the true model. Figure 3 shows results of our experiments in the coffee task. The graph shows the number of correct refinements (out of 7) detected over time, averaged over 100 trials. Time is measured as the number of actions executed, not actual computer runtime. For each tree $\mathcal{T}_i^a$, we used the parameter values $\epsilon = 0.3$, $K = 50|Dom(S_i)|$, where $Dom(S_i)$ is the domain of the state variable S_i whose conditional probabilities $\mathcal{T}_i^a$ approximates. Note that active learning outperformed passive learning and that the BIC and BDe scores performed almost identically.

In the Taxi task [17], a taxi agent has to deliver passengers from a pick-up location to their destination. In this case, the BIC and BDe scores fail to detect most of the refinements necessary to learn the true DBN model. We tested our modification to the BIC score in Equation (3) to see if regularization can improve the accuracy of the learned DBN model. Figure 4 shows results of the experiments in the Taxi task, averaged over 25 trials. In the Taxi task, the true DBN model requires 21 refinements. We used $\epsilon = 0.6$, $K = 50|Dom(S_i)|$, and report results of the BIC score for $\lambda = 0.1$ and $\lambda = 1$. The BDe score performed identically to the BIC score for $\lambda = 1$. Note that active and passive learning using the original BIC score ($\lambda = 1$) failed to detect many of the refinements of the true DBN model. With $\lambda = 0.1$, active and passive learning detected all of the refinements, with the active learning scheme being faster. We tested for values of λ between 0 and 1 in increments of 0.05, and $\lambda = 0.1$ gave the best results empirically, although we did not perform any sensitivity analysis.

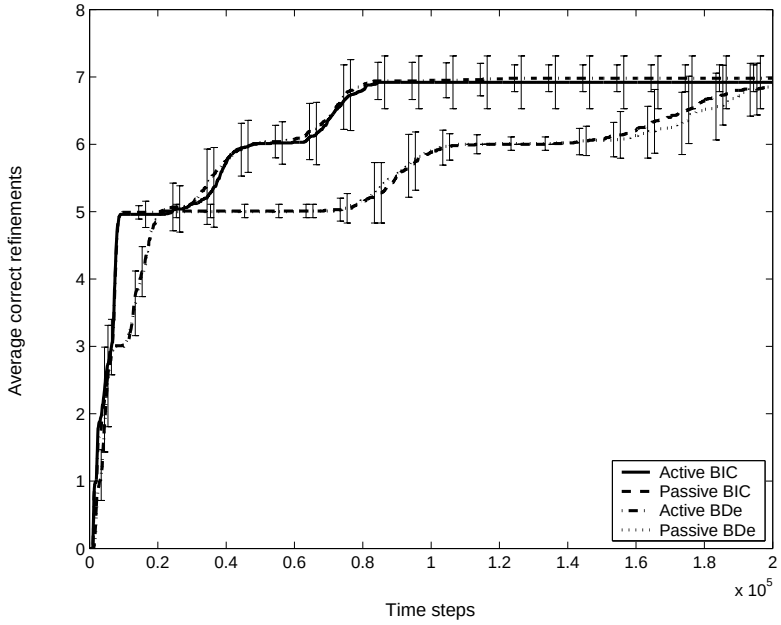

Fig. 3. Results in the coffee task

In the AGV task [18], an autonomous guided vehicle has to transport parts between machines in a manufacturing workshop. We simplified the task by reducing the number of machines to 2 and made it fully observable by setting the processing time of machines to 0. The resulting task has 75,000 states and 6 actions, and the true DBN model requires 162 refinements. Figure 4 shows results of the experiments in the AGV task, averaged over 5 trials. We used $\epsilon = 0.6$, $K = 50|Dom(S_i)|$, and report results of the BIC score for $\lambda = 0.1$ and $\lambda = 1$. There are several interesting things to notice. First, learning was very slow. We collected data for 200,000,000 time steps, and it is not clear that the graphs even converged. The learned DBN model did not come close to the true model, even for $\lambda = 0.1$. Also, passive learning actually outperformed active learning in the AGV task. We believe this is due to the fact that our active learning scheme selects actions based on local information, which we elaborate on in the conclusion. The results of the experiments in the AGV task indicate that learning DBN models of factored MDPs is a challenging problem, even using state-of-the-art metrics such as the BIC and BDe scores.

6 Conclusion

We have presented an algorithm for active learning of dynamic Bayesian networks in factored MDPs. Our approach is to learn DBNs by growing trees that represent the conditional probabilities of the DBNs. The algorithm stops to evaluate possible refinements

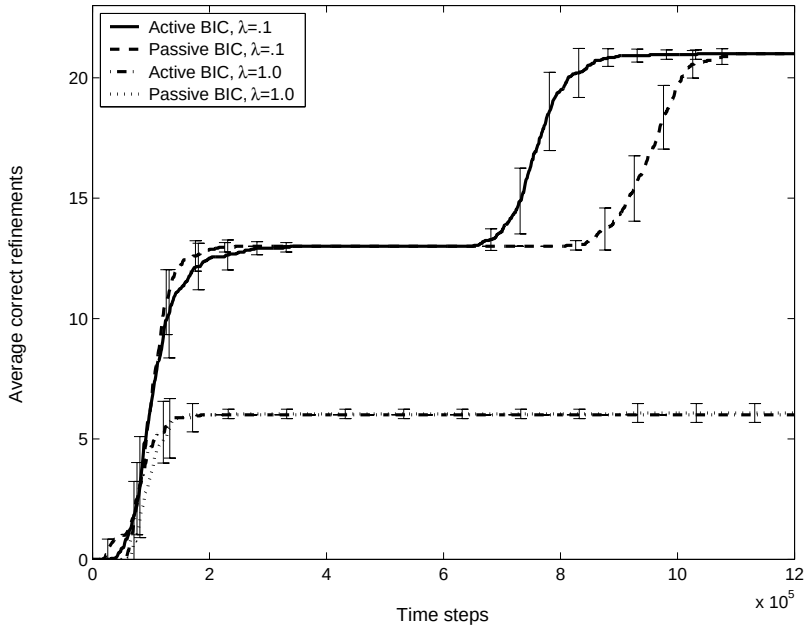

Fig. 4. Results in the Taxi task

of the trees as soon as a minimum number of data instances map to each relevant value of each potential split variable. To learn DBNs quickly, the distributions of data instances over values of each potential split variable should be uniform. We developed an active learning scheme that selects actions with the goal of maintaining the distributions as uniform as possible.

Our active learning scheme selects actions based on local information, i.e., how the distributions change locally as a result of executing actions. This works well in tasks with limited size when all states are visited relatively frequently. However, in large tasks our scheme may fail to explore large regions of the state space, prefering to maintain uniformity in the current region. We believe this accounts for the results in the AGV task. To ensure that most or all of the state space is visited it is necessary to select actions based on global information. If global information is stored using trees its size is proportional to the number of leaves of the trees, not to the number of states, facilitating scaling. Reaching a specific region of the state space is difficult when we can only sample the current trajectory since we may not know which actions will get us there. Temporally-extended actions may provide a useful tool to achieve this. Although our work is an important first step, it needs to combine with further research to achieve accurate learning of DBNs in factored MDPs.

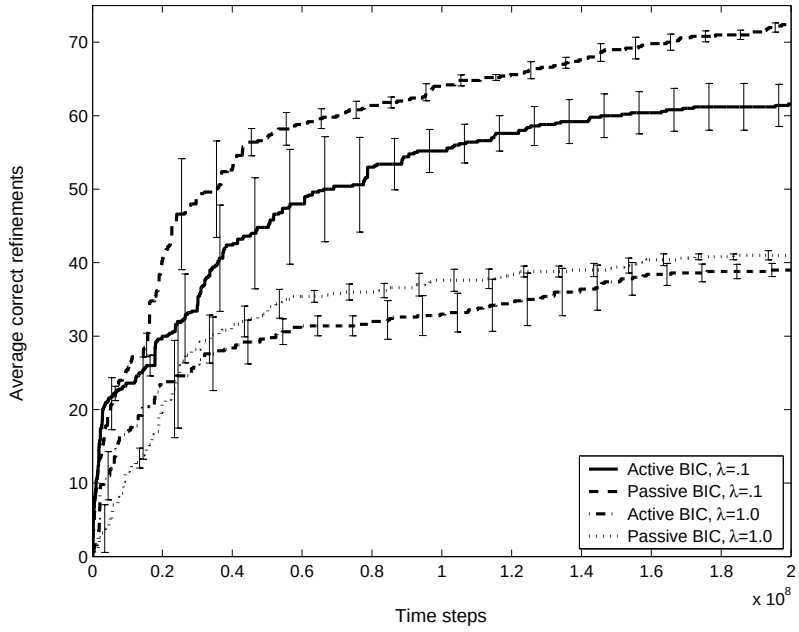

Fig. 5. Results in the AGV task

Acknowledgements This work was partially funded by NSF grants ECS-0218125 and CCF-0432143. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. Dean, T., and Kanazawa, K. (1989) A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3): 142–150.
2. Boutilier, C., Dearden, R., and Goldszmidt, M. (1995) Exploiting structure in policy construction. *Proceedings of the International Joint Conference on Artificial Intelligence*, 14: 1104–1113.
3. Feng, Z., Hansen, E., and Zilberstein, Z. (2003) Symbolic Generalization for On-line Planning. *Proceedings of Uncertainty in Artificial Intelligence*, 19: 209–216.
4. Guestrin, C., Koller, D., and Parr, R. (2001) Max-norm Projections for Factored MDPs. *Proceedings of the International Joint Conference on Artificial Intelligence*, 17: 673–680.
5. Jonsson, A., and Barto, A. (2006) Causal Graph Based Decomposition of Factored MDPs. *Journal of Machine Learning Research*, 7: 2259–2301.
6. Kearns, M., and Koller, D. (1999) Efficient Reinforcement Learning in Factored MDPs. *Proceedings of the International Joint Conference on Artificial Intelligence*, 16: 740–747.
7. Buntime, W. (1991) Theory refinement on Bayesian networks. *Proceedings of Uncertainty in Artificial Intelligence*, 7: 52–60.

8. Friedman, N., Murphy, K., and Russell, S. (1998) Learning the structure of dynamic probabilistic networks. *Proceedings of Uncertainty in Artificial Intelligence*, 14: 139–147.
9. Heckerman, D., Geiger, D., and Chickering, D. (1995) Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20: 197–243.
10. Murphy, K. (2001) *Active learning of causal Bayes net structure*. Technical report, Computer Science Division, University of Berkeley.
11. Steck, H, and Jaakkola, T. (2002) Unsupervised active learning in large domains. *Proceedings of Uncertainty in Artificial Intelligence*, 18: 469–476.
12. Tong, S., and Koller, D. (2001) Active learning for structure in Bayesian networks. *Proceedings of the International Joint Conference on Artificial Intelligence*, 17: 863–869.
13. Schwartz, G. (1978) Estimating the dimension of a model. *Annals of Statistics*, 6: 461–464.
14. Poggio, T., and Girosi, F. (1990) Regularization Algorithms for Learning that are Equivalent to Multilayer Networks. *Science*, 247: 978–982.
15. Sutton, R., and Barto, A. (1998) *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, USA.
16. Chickering, D., Geiger, D., and Heckerman, D. (1995) Learning Bayesian networks: search methods and experimental results. *Proceedings of Artificial Intelligence and Statistics*, 5: 112–128.
17. Dietterich, T. (2000) Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13: 227–303.
18. Ghavamzadeh, M., and Mahadevan, S. (2001) Continuous-Time Hierarchical Reinforcement Learning. *Proceedings of the International Conference on Machine Learning*, 18: 186–193.