

Exploiting Belief Locality in Run-Time Decision Theoretic Planners

William H. Turkett, Jr. **John R. Rose**

Department of Computer Science
Wake Forest University
Winston Salem, NC 27104
turketwh@wfu.edu

Department of Computer Science
University of South Carolina
Columbia, SC 29208
rose@cse.sc.edu

Abstract

While Partially-Observable Markov Decision Processes have become a popular mechanism for representing realistic planning problems, exact approaches to finding POMDP solutions are feasible only for small domains due to their computational complexity. One of the key problems for traditional approaches is their requirement to work with the entire state space, even though a planner is likely to visit only a small part of that space during any execution period. An alternative approach for control in POMDP domains is to use run-time optimization over action sequences in a dynamic decision network. While not guaranteed to be optimal, this run-time approach reasons only about the reachable parts of the belief space. By combining a runtime planning approach with caching mechanisms, it is possible to exploit belief locality in POMDP domains, allowing for significant speedups in plan generation times. This paper introduces and evaluates an exact caching mechanism and a grid-based caching mechanism.

Introduction

In recent years, autonomous systems have become increasingly popular, both as an area of research and in practical application. The domains that these systems are applied in are becoming increasingly more complex, and will continue to do so. One of the key contributors to such complexity is the element of uncertainty present in many domains. Two types of uncertainty that must be addressed are action uncertainty, where only a distribution over possible action outcomes is known, and state uncertainty, where an agent's observations only provide partial information about the true state of the world. Partially-Observable Markov Decision Processes (POMDPs) have become a standard mechanism for representing such domains, as the POMDP formalization can represent both of these types of uncertainty, as well as model cost and goal functions. When situated in a POMDP environment, an agent wishes to select the optimal action to perform to achieve its goals, given its current beliefs.

The traditional approach to solving POMDPs is for an agent to find a complete policy, the belief to action mapping, for a domain before execution. Exact and optimal algorithms have been defined for finding these policies, but such algorithms are extremely computationally complex. Two particular causes of this complexity are the high dimensional, continuous belief space seen in many POMDP domains, and the exponential number of action/observation sequences an agent must consider as it reasons about the future. Accordingly, a significant amount of current research has focused on approximate algorithms capable of finding near-optimal policies under significantly reduced computational requirements. While many of these algorithms maintain the approach of pre-execution policy generation, it is also possible to use run-time and simulation-based algorithms. In these latter approaches, an agent focuses on predicting forward from its current belief state and just considering reachable sets of states. Such an approach can avoid parts of the complexity caused by the enormity of the belief space. Whether these algorithms are being used at run-time to select actions or are being used in a simulation mode to generate partial policies, they need to be computationally efficient to make planning feasible and preserve agent reactivity.

Our research suggests that, across a variety of different domains, there tends to be a significant amount of locality in the belief states an agent enters during execution when given an initial start state. If this belief locality can be exploited, significant speedups for run-time based approaches are possible. In particular, it is believed that there are two types of belief locality that can be exploited: 1) common entry into a small set of belief states and 2) common entry into a few clusters of belief states, where belief states within a cluster are highly similar. Belief locality of the first type is extremely useful in allowing speedups and is most common when actions are near-deterministic and observations are informative (intuitively, the problems for which some form of domain exploitation should produce the most speedup). This paper presents and evaluates algorithms for exploiting both types of belief

locality within a run-time based POMDP planning architecture.

Background

Partially-Observable Markov Decision Processes allow for representations of problems where there is both action uncertainty and state uncertainty. Modeling action uncertainty requires maintaining a table of possible action outcomes for each possible true world state and action. Modeling observation uncertainty requires a table of likelihoods for potential observations that an agent may receive, given the agent's current state and/or the action just performed. An agent represents its state uncertainty with a vector of likelihoods representing the probability of being in each possible true world state. A given instance of this vector is called a belief state, and the set of possible belief states is the belief space. Given a current belief state, an action that has just been executed, and the corresponding observation, an agent can determine its next belief state with minimal computation (the problem of updating belief states is Markovian in POMDPs). However, if one is planning, an actual observation cannot be made. Instead, a probability distribution over observations is given after selection of an action. This probability distribution over observations forces a probability distribution over the agent's potential new belief state.

Because the problem of updating belief states is Markovian in POMDPs, the POMDP planning problem can be cast as the simpler MDP problem defined over a continuous belief space. This MDP can then be solved by a modified value iteration algorithm. This is the approach taken by traditional, exact POMDP solvers. Instead of representing the value function as a table of states and values as for MDPs, the value function is represented as a piecewise, linear convex function over the entire belief space.

Because POMDP models are probability distributions, the dynamics of POMDP environment can also be represented by a Bayesian network. Bayesian networks are a graphical data structure that allow for factored, variable-oriented representations of probability distributions. Please see the work of Finn Jensen (Jensen, 2001) for a complete definition of Bayesian networks. Given a Bayesian network, evidence can be entered on the random variables. The structure of the graph is then used in propagating evidence through the variables and determining the probability of being in various states (Jensen, 2001). Temporal Bayesian Networks can be used to represent progressive states of a system over distinct time-slices. The initial time-slice represents the initial state of the system and edges between time-slices represent transitional probabilities for variables that are dependent across time-slices (Boutilier, 1999).

The standard Bayesian network can also be used as a basis for an influence diagram, a structure that incorporates decision and utility nodes along with random variables. Decision nodes represent actions that can be taken, while utility nodes represent rewards and costs conditioned on variable states and actions. If evidence is entered into an influence diagram, the expected utility of performing each action in the decision node can be calculated and used to determine the highest expected utility action to perform (Boutilier, 1999).

Dynamic decision networks extend general influence diagrams over a finite planning horizon, much like temporal Bayesian networks. The state of the agent and a decision node exists at each time slice to represent the agent's state and the action to be performed in that slice. Utilities can be represented in each time slice if the reward is time-separable or at the end of the sequence of actions. By selecting an action in the initial time step, the predicted states resulting from performance of that action can then be used in evaluating the actions available in the next time step. The solution algorithm for DDNs finds the maximum expected utility sequence of actions (Russell, 2002) (Forbes, 1995).

The underlying algorithm for determining the optimal sequence of actions to perform relies on the process of variable elimination in the decision network (Jensen, 2001) [derived from the corresponding Bayesian network variable elimination algorithm]. Beginning with the last decision, variable elimination in DDNs finds a function for each decision which, given any past, determines the maximum utility action to take. The complexity of the potential computations is dependent on the largest set of nodes that is needed during this process. For general dynamic decision networks, this largest set of nodes contains all action nodes and observation nodes present in the network, as well as single nodes for the belief variables. The fact that dynamic decision networks algorithms are optimizing over possible sequences of actions and observations is seen in the composition of this largest set of nodes.

Methodology

The run-time POMDP planning architecture that has been developed uses DDNs as the central planning component within a traditional sense-plan-act control mechanism. To plan, the agent's current beliefs about the true state of the world are set in the variables in the first time-step of the dynamic decision network. By propagating these beliefs through the network and maximizing over action sequences, an optimal sequence of actions (of length equal to the number of time-steps represented in the network) is returned. The execution system then executes the first action returned from the planning component. Once this

action has been executed, the agent receives new observations about the world that are used to update its belief state. Since the optimal sequence of actions previously generated by the planner may no longer be appropriate, re-planning is done to select the new next best action to perform.

Caching

For a given problem, the total number of possible belief states is infinite, ranging over all possible assignments to the true states of the world of the sets of real numbers between zero and one that sum to one. Many of these belief states, however, will never be encountered during execution by an agent in a given domain. In fact, many of the states may not even be reachable from a given start state. In these situations, finding a complete policy beforehand is less justified and run-time approaches become more appealing. When environments are near-deterministic, there are trajectories of belief states that the agent will encounter fairly often. An agent planning at runtime may be able to discover and take significant advantage of this knowledge about which belief states are important in its planning process. In many areas of computer science, such as processor architecture and web storage, caches are used to exploit these types of locality, which suggests that a cache based approach may also allow for improvements in the agent planning context when there is locality in belief states.

The implementation of caches in the developed planning architecture performs a cache check prior to performing any DDN action selection. If a cache entry is found for the present belief state, the appropriate action stored in the cache is sent directly to the execution mechanism and the DDN planning mechanism is never used. Updates to the cache, which requires storage of a (belief state, action) pair are performed for the current belief state whenever the DDN planning mechanism is used. In the developed architecture, cache entries are preserved across multiple runs of an agent, allowing the agent to make use of all of its past experiences.

In one sense, caching optimal actions at runtime is essentially the process of building a partial policy, where only the interesting parts of the policy (interesting to the agent in its current context) are constructed. When computation times are still too expensive to allow for true run-time planning, an agent designer could allow the agent to simulate a large portion of the state space prior to execution, storing its chosen actions during the simulation in the cache. This cache can then be used as the basis for a reactive policy, with the appropriate actions for the states that the agent is likely to enter at runtime already pre-computed.

Exact Caching The simplest form of cache to implement and use is a traditional, exact match cache. To implement a traditional cache, a hash table is used for storing a representation of a belief state and the corresponding

optimal action. The hash key for the cache table is a string that represents the factored belief state of the agent. By using the factored belief state representation, the key can be relatively small compared to the true dimensionality of the state space. This factored representation is also inexpensive to generate due to the manner in which the agent's beliefs are stored. Since the agents are using decision networks for planning, the natural underlying belief representation is the state of the factored variables. These beliefs can then be easily mapped to the variables in the initial time-slice of the DDN used for planning and concatenated to generate a unique belief state hash key.

In many proposed domains for autonomous entities, there will often be limits on the available memory. Accordingly, there will likely be limits on the cache size that an agent can make use of. An agent in this situation needs to use an intelligent replacement algorithm to ensure that it does not invalidate commonly used or soon to be used again cache entries. In the current implementation, cache entries are replaced using the least recently used (LRU) replacement algorithm. A timestamp is stored when a belief state is either added to the cache or accessed to support the LRU algorithm.

Grid Caching A second approach to using caches in POMDP planning is to simulate grid-based POMDP approximation algorithms. In grid-based POMDP algorithms, optimal actions are pre-computed for different points in the belief space (the points belonging to the grid). Action selection is implemented by interpolating to find the closest pre-computed belief point and performing the action optimal for the close point. An example of recent advances in grid-based POMDP approximation algorithms can be found in Pineau's point-based value iteration approach (Pineau, 2003). Ideas very similar to the POMDP grid-approximation work can be used in combination with caching and run-time planning, allowing the agent to build a fine grid around the interesting and reachable parts of the state space.

The implemented grid-based POMDP cache uses a belief similarity retrieval scheme. In this scheme, the agent designer sets a threshold for the maximum difference between the agent's current belief state and a cached belief state the agent will allow when selecting actions. The smallest threshold would be zero, wherein the cache is equivalent to the standard exact-match cache. Thresholds greater than zero, while potentially useful for approximating the best action, also have the potential to degrade the performance of the agent as the use of an action optimal for the closest belief point may not always be optimal for the current belief point. The current implementation of the grid-based cache defines the difference in belief states as the cumulative shift in the probability distribution (the Manhattan distance between the two belief points). For two belief states, this shift is computed as the sum of the difference in likelihoods over all underlying true states. This approach was chosen because it is fast to compute and can be stopped once the

acceptable threshold distance between the belief states has been passed. Other measures, such as Euclidean distance or KL divergence between the two belief points, could also be made use of in this context.

If the difference between the current belief state and one in the cache is below the designer's threshold, the agent is willing to accept the cached action for the nearby state as the best action for the current state. If two or more points are below the threshold, the closest point is used as the approximating belief point. This approach grows computationally more expensive as the cache grows, as the agent needs to compute the distance between each belief point in the cache and the current belief point. If no point is close enough to the current belief state, the agent uses its DDN to generate an optimal action.

The replacement algorithm for a grid-based cache is still implemented as least-recently-used, and the timestamp on the belief state present in the cache that is most similar to the current belief state is updated to reflect its usage. If the cache is used to select an action, the true belief state is not added to the cache to ensure that the incremental propagation of small belief differences does not introduce significant error. Only (belief-state, action) pairs generated directly from the agents DDN are added to the cache.

For the grid-based cache, the semantics of a shift in belief space are based on differences between true belief states. Accordingly, the comparisons can not be performed between vectors representing factored state entries. Since a representation of the non-factored belief state is used for the cache key, a considerably larger amount of space is required with grid caching than with exact caching. An alternative would be to store in the cache the factored representation and then compute the non-factored belief state for every cache entry when the cache is searched. Since caches can grow fairly large, the current implementation takes advantage of available memory to prevent the conversion from factored to un-factored belief representation during cache searches.

Implementation Details

The current implementation of the agent architecture described in this paper is based on Zeus and Hugin. Zeus is a generic agent architecture in Java. Hugin is a Bayesian Network API with a Java interface. Simulations used to test the implemented agent architecture were performed on a Sun dual 750MHz processor Sun-Fire-280R with 4 gigabytes of RAM, with the agent architecture allocated one gigabyte of RAM for internal use. The dual processors of the machine were controlled only by the OS and were used only in handling different processes. No parallelization of any agent tasks was performed.

Evaluation

The performance of the exact and grid cache run-time planning algorithms has been examined on a set of six varied problems from the current POMDP literature. Due to space limitations, a subset of the results for three problems will be presented here. Figure 1 demonstrates the usefulness of exact caching on the Large Cheese Taxi problem, while Figures 2 and 3 and Tables 1 and 2 presents results from using exact and grid caching on the Hallway and Robot Tag problems. Please refer to previous work of the author (Author, 2004) for a more detailed description of the problems used for testing and further results related to exact and grid based caching.

The three problems presented have state sizes of 60, 300, and 870, all of which are significantly larger than those solvable by exact POMDP planners. The initial start state for the problems of interest in this work is the no-information state, where the agent has uniform beliefs about its current state in the world.

The first problem presented is a large taxi-navigation problem. It is an extension of the cheese taxi problem from the POMDP literature (Pineau, 2002). In this domain, the agent can be in one of thirty locations, labeled 0 through 29, and have one of ten destinations, a depot and nine scattered drop-off locations, for a total of 300 states. The agent can perform seven actions: North, South, East, West, Pickup, Putdown, and Query. If the agent has a depot destination, its task is to Pickup the passenger from a specific depot state. The agent should then navigate the passenger to his requested destination and Putdown the passenger. The problem is made more difficult, however, as the passenger does not mention where he would like to go unless a Query action is performed. Also, there is a 5% chance the passenger will change his destination choice as the taxi is navigating through a small set of locations. The passenger must be queried for the new destination if the destination changes. Appropriate rewards are earned for a correct Putdown and appropriate costs are used for navigation and incorrect Putdowns.

The graph in Figure 1 shows the comparison between the planning times required across 250 trials for an agent without caching and an agent using an exact cache with a maximum of 500 entries. A trial consists of multiple actions and is complete when the agent achieves the goal for the domain. While this domain has 300 states, those states are easily learned by the agent. Note that at around action 300, the median planning time for the agent has dropped close to zero. This suggests that the belief space itself (those distributions over possible true world states) is not very large and is consistently re-visited.

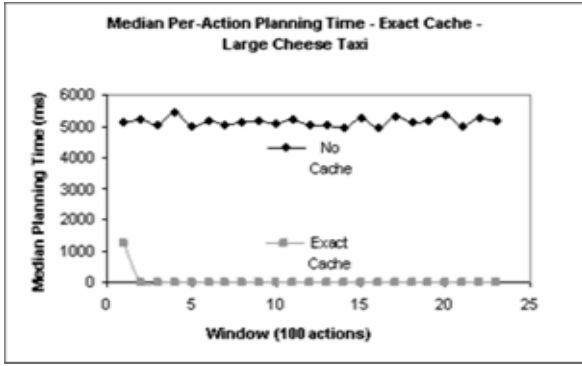


Figure 1. Median Planning Times for Large Cheese Taxi Problem

The Hallway problem was originally introduced by Littman, Cassandra, and Kaelbling (Littman, 1995). In this problem, an agent must navigate through a series of rooms to a specified goal state. The agent can perform five different actions: Stay, MoveForward, TurnLeft, TurnRight, and TurnAround. There are 60 true states the agent can be in: any of four directions within 15 possible different rooms. Transitions are noisy, leading the agent to end up in the wrong direction or wrong room fairly often. Observations are also noisy, with many observations possible for each state. When in three specific rooms facing south and when in the goal state, the agent is guaranteed to receive a landmark observation which gives the agent knowledge of its exact location.

The graph in Figure 2 shows the median planning times required by an agent over 200 trials when planning with no cache, with an exact cache of unlimited size, and with grid-caches of unlimited size and five different threshold values. The top set of data points in the graph are the no-cache results, the middle set of points represent the exact cache results, and the four clustered sets of points at the bottom are the grid-cache results. Exact caching shows an improvement in performance over no cache, but the improvement is not as consistent as was seen in the previous problem.

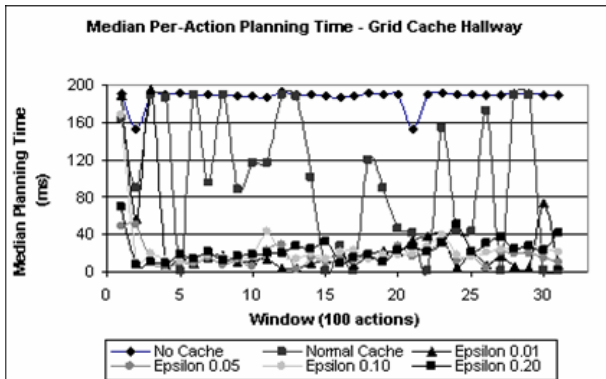


Figure 2. Median Planning Time for Hallway Problem

| Cache Threshold | Average Reward |
|-----------------|----------------|
| 0.00 | -8.30 |
| 0.01 | -8.23 |
| 0.05 | -8.39 |
| 0.10 | -8.72 |
| 0.20 | -9.03 |

Table 1. Cache Threshold vs. Average Reward Comparison for Hallway Problem

The Hallway problem is noisy in observations and in action outcomes and thus a wider range of belief states is entered and belief states are revisited less often.

After approximately 200 actions, however, the grid cache approaches have seen enough of the belief space to allow rapid extraction of useful actions. This supports the idea that, although the agent is entering many more belief states due to the domain dynamics, many of these belief states are very similar to each other and have the same optimal action definition. Since a grid-cache approach approximates the optimal action for the current state by using results from nearby states, there is the potential for a loss in policy quality. Policy quality can be measured by the total average reward earned over a set of trials. Table 1 presents the average rewards that correspond to the thresholds and planning times presented in Figure 1. The rewards earned are very close, particularly when the thresholds are small. Note that a threshold of 0.00 indicates use of the exact cache, where there is no grid-based approximation. A Mann-Whitney comparison of the rewards returns p-values > 0.05 , meaning that it is not possible to reject the null hypothesis that these rewards are from the same distribution.

The robot laser tag problem is another problem from Pineau (Pineau, 2003). In this problem, a robot player can be in one of 29 positions on a board and the opponent can be in one of the same 29 positions or in the Tagged state, providing for an 870 state problem. The player can move North, South, East, West, as well as Tag the opponent. All actions are deterministic in that the robot will move correctly where it desires to move. The location of the robot is fully observable, while the location of the opponent is unobservable until the robot and opponent are in the same location. There are thirty possible observations - a "same position" observation returned when the robot and opponent are in the same location, and the robot location for all other states of the world.

The graph for this problem, shown in Figure 3, represents performance over 250 trials and is similar to that for the hallway problem, showing improvements when using the exact cache and larger and faster improvements when using the grid-cache. Average rewards earned across trials for each grid threshold level are presented in Table 2. Rewards are preserved across grid threshold levels, as a Mann-Whitney test returns p-values > 0.05 , suggesting that it is not possible to say these rewards are not from the same distribution. This problem does begin to show the

extra costs required by a grid caching algorithm. As an agent adds more to its cache, there is an increased cost to find the minimum distance cache entry. A significant upward trend in median per-action planning times is seen at the right end of the graph in Figure 3. A similar trend is also present in the previous hallway graph.

| Cache Threshold | Average Reward |
|-----------------|----------------|
| 0.00 | -15.55 |
| 0.01 | -15.75 |
| 0.05 | -16.16 |
| 0.10 | -16.43 |
| 0.20 | -15.16 |

Table 2. Cache Threshold vs. Average Reward Comparison for Robot Tag Problem

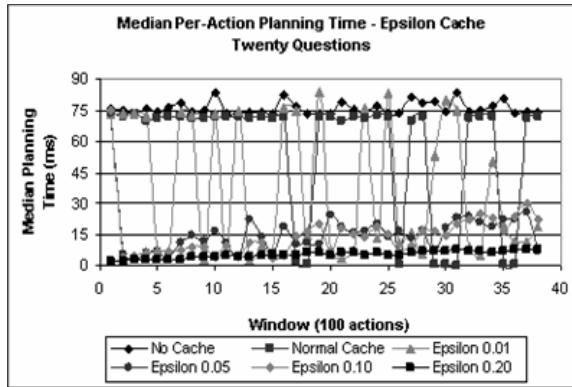


Figure 3. Median Planning Times for Robot Tag Problem

Conclusions

This work and previous work by the author (Author 2004) suggest that, across a variety of POMDP domains, there is belief locality which can be exploited by a run-time planning agent. In particular, caching of previous results appears to perform very well on near-deterministic and informative domain. We believe this work also demonstrates that grid-based caching is an effective alternative over exact caching for domains that are noisy and have a large number of possible action outcomes. On the set of problems tested, grid based caches worked well, showing significant reductions in computation time with minimal loss in earned average reward. This suggests that, even in difficult domains, the sets of belief states that are entered are still clustered closely together. Consider an agent that enters a true world state it has been in before in the same goal context as before - the agent is likely be in a belief state that is very similar to the one it held previously in that true world state.

There are three areas of future work that stem directly from the work presented here. The first improvement that can be made is to seed the cache with the fully-observable MDP policy. The FOMDP policy is relatively cheap to

compute (polynomial). This should prevent the agent from using the DDN to compute an action when the optimal action can just be taken from the FOMDP policy. This may also aid the agent in grid-cache searching around the FOMDP belief points. A second area of interest is in determining optimal training runs to be used for building partial policies. If caching is being used to build a partial policy before execution, some training paths will allow coverage of the domain much faster than others. If an agent is trained moving to the farthest passenger destinations first in the Large Cheese Taxi domain, it is likely that the desired partial policy can be built much faster than if the destinations are chosen at random, as the agent will encounter large numbers of states early on. The third area of interest for future work is in improving the computational requirements for searching the grid-based cache. A data structure similar to a metric-tree, whose properties ensure only part of the tree has to be searched at any time, should show performance improvements for the grid cache.

References

- Boutilier, Craig; Dean, Thomas; and Hanks, Steve. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11:1-94.
- Forbes, Jeff; Huang, Tim; Kanazawa, Keiji; and Russell, Stewart. 1995. The Batmobile: Towards a Bayesian Automated Taxi. In *Proceedings of the Fourteenth Joint Conference on Artificial Intelligence*.
- Jensen, Finn. 2001. *Bayesian Networks and Decision Graphs*. New York, NY: Springer-Verlag.
- Littman, Michael L.; Cassandra, Anthony; and Kaelbling, Leslie Pack. 1995. Learning Policies for Partially Observable Environments: Scaling Up. In *Proceedings of the Twelfth International Conference on Machine Learning*. 394-402.
- Pineau, Joelle and Thrun, Sebastian. 2002. An Integrated Approach to Hierarchy and Abstraction for POMDPs. Technical Report, CMU-RI-TR-02-21, Department of Computer Science, Carnegie Mello University.
- Pineau, Joelle; Gordon, Geoff; and Thrun, Sebastian. 2003. Point-Based Value Iteration: An Anytime Algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Russell, Stewart and Norvig, Peter. 2002. *Artificial Intelligence: A Modern Approach*. 2nd edition. Englewood Cliffs, NJ: Prentice Hall.
- Author self reference. 2004. Robust Multiagent Plan Generation and Execution with Decision-Theoretic Planners. Ph.D.

Dissertation, Department of Computer Science and Engineering,
Author University Self-Reference.