Symbolic Heuristic Search for Probabilistic Planning

Zhengzhu Feng
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
fengzz@cs.umass.edu

Eric A. Hansen
Department of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39762
hansen@cse.msstate.edu

Abstract

We describe a planner that participates in the Probabilistic Planning Track of the 2004 International Planning Competition. Our planner integrates two approaches to solving Markov decision processes with large state spaces. State abstraction is used to avoid evaluating states individually. Forward search from a start state, guided by an admissible heuristic, is used to avoid evaluating all states.

Introduction

The 2004 International Planning Competition introduces, for the first time, a probabilistic planning track. The underlining model of the planning problem is essentially a Markov decision process (MDP), and is encoded using an extension of the PDDL language, called the Probabilistic PDDL. Classic dynamic programming algorithms solve MDPs in time polynomial in the size of the state space. However, the size of the state space grows exponentially with the number of features describing the problem. This “state explosion” problem limits use of the MDP framework, and overcoming it has become an important topic of research.

Over the past several years, approaches to solving MDPs that do not rely on complete state enumeration have been developed. One approach exploits a feature-based (or factored) representation of an MDP to create state abstractions that allow the problem to be represented and solved more efficiently (Dearden & Boutilier 1997; Hoey et al. 1999; and many others). Another approach limits computation to states that are reachable from the starting state(s) of the MDP (Barto, Bradtke, & Singh 1995; Dean *et al.* 1995; Hansen & Zilberstein 2001). Our planner integrates these approaches in a unifying framework using symbolic model-checking techniques, based on the symbolic LAO* and symbolic RTDP algorithms we previously developed (Feng & Hansen 2002; Feng, Hansen, & Zilberstein 2003). In this paper we present a brief summary of these algorithms.

Factored MDPs and decision diagrams

A Markov decision process (MDP) is defined as a tuple (S, A, P, R) where: S is a set of states; A is a set of actions; P is a set of transition models $P^a : S \times S \rightarrow [0, 1]$, one for each action, specifying the transition probabilities of the process; and R is a set of reward models $R^a : S \rightarrow \Re$, one for each action, specifying the expected reward for taking action a in each state. We consider MDPs for which the objective is to find a policy $\pi : S \rightarrow A$ that maximizes total discounted reward over an infinite (or indefinite) horizon, where $\gamma \in [0, 1]$ is the discount factor. (We allow a discount factor of 1 for indefinite-horizon problems only, that is, for MDPs that terminate after a goal state is reached.)

In a factored MDP, the set of states is described by a set of random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. Without loss of generality, we assume these are Boolean variables. A particular instantiation of the variables corresponds to a unique state. Because the set of states $S = 2^{\mathbf{X}}$ grows exponentially with the number of variables, it is impractical to represent the transition and reward models explicitly as matrices when the number of states variables is large. Instead we follow Hoey *et al.*(1999) in using algebraic decision diagrams to achieve a more compact representation.

Algebraic decision diagrams (ADDs) are a generalization of binary decision diagrams (BDDs), a compact data structure for Boolean functions used in symbolic model checking. A decision diagram is a data structure (corresponding to an acyclic directed graph) that compactly represents a mapping from a set of Boolean state variables to a set of values. A BDD represents a mapping to the values 0 or 1. An ADD represents a mapping to any finite set of values. To represent these mappings compactly, decision diagrams exploit the fact that many instantiations of the state variables map to the same value. In other words, decision diagrams exploit state abstraction. BDDs are typically used to represent the characteristic functions of sets of states and the transition functions of finite-state automata. ADDs can represent weighted finite-state automata, where the weights correspond to transition probabilities or rewards, and thus are an ideal representation for MDPs.

Hoey *et al.* (1999) describe how to represent the transition and reward models of a factored MDP compactly using ADDs. We adopt their notation and refer to their paper for details of this representation. Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ represent the state variables at the current time and let $\mathbf{X}' = \{X'_1, \ldots, X'_n\}$ represent the state variables at the next step. For each action, an ADD $P^a(\mathbf{X}, \mathbf{X}')$ represents the transition probabilities for the action. Similarly, the reward model $R^a(\mathbf{X})$ for each action a is represented by an ADD. The advantage of using ADDs to represent mappings from states (and state transitions) to values is that the complexity of operators on ADDs depends on the number of nodes in the diagrams, not the size of the state space. If there is sufficient regularity in the model, ADDs can be very compact, allowing problems with large state spaces to be represented and solved efficiently.

Symbolic LAO* algorithm

LAO* (Hansen & Zilberstein 2001) is an extension of the classic search algorithm AO* that can find solutions with loops. This makes it possible for LAO* to solve MDPs, since a policy for an infinite-horizon MDP allows both conditional and cyclic behavior. Like AO*, LAO* has two alternating phases. First, it expands the best partial solution (or policy) and evaluates the states on its fringe using an admissible heuristic function. Then it performs dynamic programming on the states visited by the best partial solution, to update their values and possibly revise the currently best partial solution. The two phases alternate until a complete solution is found, which is guaranteed to be optimal.

AO* and LAO* differ in the algorithms they use in the dynamic programming step. Because AO* assumes an acyclic solution, it can perform dynamic programming in a single backward pass from the states on the fringe of the solution to the start state. Because LAO* allows solutions with cycles, it relies on an iterative dynamic programming algorithm (such as value iteration or policy iteration). In organization, the LAO* algorithm is similar to the "envelope" dynamic programming approach to solving MDPs (Dean *et al.* 1995). It is also closely related to RTDP (Barto, Bradtke, & Singh 1995), which is an on-line (or "real time") search algorithm for MDPs, in contrast to LAO*, which is an off-line search algorithm.

We call our generalization of LAO* a symbolic search algorithm because it manipulates sets of states, instead of individual states. In keeping with the symbolic model-checking approach, we represent a set of states S by its characteristic function χ_S, so that $s \in S \iff \chi_S(s) = 1$. We represent the characteristic function of a set of states by an ADD. (Because its values are 0 or 1, we can also represent a characteristic function by a BDD.) From now on, whenever we refer to a set of states, S, we implicitly refer to its characteristic function, as represented by a decision diagram.

In addition to representing sets of states as ADDs, we represent every element manipulated by the LAO* algorithm as an ADD, including: the transition and reward models; the policy $\pi : S \rightarrow A$; the state evaluation function $V : S \rightarrow \Re$ that is computed in the course of finding a policy; and an admissible heuristic evaluation function $h : S \rightarrow \Re$ that guides the search for the best policy. Even the discount factor γ is represented by a simple ADD that maps every input to a constant value. This allows us to perform all computations of the LAO* algorithm using ADDs.

Besides exploiting state abstraction, we want to limit computation to the set of states that are reachable from the start state by following the best policy. Although an ADD effectively assigns a value to every state, these values are only relevant for the set of reachable states. To focus computation on the relevant states, we introduce the notion of *masking* an ADD. Given an ADD D and a set of relevant states U, masking is performed by multiplying D by χ_U. This has the effect of mapping all irrelevant states to the value zero. We let D_U denote the resulting *masked ADD*. (Note that we need to have U in order to correctly interpret D_U). Mapping all irrelevant states to zero can simplify the ADD considerably. If the set of reachable states is small, the masked ADD often has dramatically fewer nodes. This in turn can dramatically improve the efficiency of computation using ADDs.

Symbolic LAO* does not maintain an explicit search graph. It is sufficient to keep track of the set of states that have been "expanded" so far, denoted G, the *partial value function*, denoted V_G, and a *partial policy*, denoted π_G. For any state in G, we can "query" the policy to determine its associated action, and compute its successor states. Thus, the graph structure is implicit in this representation. Note that throughout the whole LAO* algorithm, we only maintain one value function V and one policy π. V_G and π_G are implicitly defined by G and the masking operation.

Symbolic RTDP

Recall that RTDP performs a DP update while interacting with the environment. At each time step t, the agent observes the current state s_t and performs a DP backup to update its value, as follows:

$$V^{t+1}(s_t) \leftarrow \max_{a \in A} \left\{ R^a(s_t) + \gamma \sum_{s' \in S} P^a(s_t, s') V^t(s') \right\}. \tag{1}$$

The values of all other states are kept unchanged, that is, for all $s \neq s_t$:

$$V^{t+1}(s) = V^t(s).$$

If the initial value function is an admissible estimate of the optimal value function, then an agent can always take the action that maximizes Equation (1). Otherwise some exploration scheme must be used in choosing actions, in order to ensure convergence. After an action is taken, the agent observes the resulting state and the cycle repeats.

The advantage of RTDP over standard DP is that it uses an on-line trajectory of states, beginning from the start state, to determine what states to update and to avoid computations on unlikely states. However, the enumerative nature of the trajectory sampling is a bottleneck for further performance improvement. When the state space is large enough, a state by state update becomes hopelessly inefficient, especially if the sampling involves carrying out physical actions. sRTDP helps overcome this inefficiency by generalizing the update from a single state to an abstract state, using symbolic model checking techniques.

We extend the idea of masking in symbolic LAO* to sRTDP by performing DP on the abstract state E that the current state s belongs to. Symbolic model-checking provides us with convenient and efficient techniques to group states as abstract states and to manipulate these abstract states. There are many ways to group states into abstract states. We present two heuristic approaches that are motivated by the idea of generalization by structural similarity. A *value-based* abstract state consists of states whose value estimates are close to that of the current state. A *reachability-based* abstract state consists of states that share with the current state a similar set of successor states. Unlike SPUDD, we *explicitly* construct this abstract state at each time step of sRTDP, using standard ADD model-checking operators.

Generalization by Value With a value-based abstract state, the experience is generalized to states that have similar value estimates as the current state. The intuition is that states with similar *optimal* values may also be similarly desirable. Generalizing updates to states with similar *estimated* values helps the agent in two ways. First, if some of these states indeed have similar optimal value as the current state, the update strengthens this similarity and the agent is better informed in the future when these states are visited again. Second, if some of the states have very different optimal value than the current state, the generalization helps to distinguish them and avoid computations on them in the future when the same state as the current state is visited again.

Generalization by Reachability With a reachability-based abstract state, the experience is generalized to states that are similar to the current state in terms of the set of one-step reachable states. The intuition here is that if the agent is going to visit some states, say C, from the current state s, then any information about C is useful not only to s but also to other states that can reach C. By generalizing the update to these other states the agent is better informed in the future whether to aim at C or to avoid it.

To compute the abstract state based on reachability, we introduce two operators from the model-checking literature. The $Img(C)$ operator computes the set of one-step reachable states from states in C, and the $PreImg(C)$ operator computes the set of states that can reach some state in C in one step. The reachability-based abstract state E can then be computed as:

$$E = PreImg(Img(\{s\})) - PreImg(S - Img(\{s\})).$$

Once the set E is computed, it is used to mask the current value function before perform the DP update. After the update, an action is chosen that maximizes the DP update at state s. The agent then carries out the action, and the process repeats.

Although both symbolic LAO* and sRTDP use a "masked" DP update, the masks they use are different and serve different purposes. The mask in symbolic LAO* contains all states visited so far by the forward search step. The purpose of masking is to restrict computation to relevant states. The mask in sRTDP contains states that share structural similarity. The purpose of masking is to generalize update on a single state to an abstract state. This generalization has two consequences. It introduces some overhead in the DP step, including identifying the abstract state, and preforming masked DP instead of single-state DP. On the other hand, it updates the value of a group of states in a single step, at a cost that can be significantly less than updating the states separately. For problems that are large enough yet have special

Admissible heuristics

Both LAO* and (model-based) RTDP use an admissible heuristic to guide the search. From the initial release of the sample test problems from the planning competition, it is possible to design domain specific heuristic functions. On the other hand, if such a heuristic is not available, we can always revert to a simple heuristic using approximate dynamic programming. Given an error bound on the approximation, the value function can be converted to an admissible heuristic. (Another way to ensure admissibility is to perform value iteration on an initial value function that is admissible, since each step of value iteration preserves admissibility.) Symbolic dynamic programming can be used to compute an approximate value function efficiently. St. Aubin et al. (2000) describe an approximate dynamic programming algorithm for factored MDPs, called APRICODD, that is based on SPUDD. It simplifies the value function ADD by aggregating states with similar values. Another approach to approximate dynamic programming for factored MDPs described by Dearden and Boutilier (1997) can also be used to compute admissible heuristics.

References

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *Artificial Intelligence* 76:35–74.

Dearden, R., and Boutilier, C. 1997. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89:219–283.

Feng, Z., and Hansen, E. A. 2002. Symbolic heuristic search for factored markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*.

Feng, Z.; Hansen, E. A.; and Zilberstein, S. 2003. Symbolic generalization for on-line planning. In *Proceedings of the 19th Conference on Uncertainty in Articial Intelligence*.

Hansen, E., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Articial Intelligence*, 279–288.

St-Aubin, R.; Hoey, J.; and Boutilier, C. 2000. APRICODD: Approximate policy construction using decision diagrams. In *Proceedings of NIPS-2000*.