Faster Dynamic Programming for Markov Decision Processes

Peng Dai and Judy Goldsmith
Computer Science Dept.
University of Kentucky
Lexington, KY 40506-0046
daipeng@uky.edu

Abstract

Markov decision processes (MDPs) are a general framework used in artificial intelligence (AI) to model decision theoretic planning problems. Solving real world MDPs has been a major and challenging research topic in the AI literature, since classical dynamic programming algorithms converge slowly. We discuss two approaches in expediting dynamic programming. The first approach combines heuristic search strategies with dynamic programming to expedite the convergence process. The second makes use of graphical structures in MDPs to perform dynamic programming in a better order.

Introduction

The problem of decision theoretic planning has become a central research topic in AI, not only because it is an extension to classical planning, but also due to its close connection with solving real world problems. Markov decision processes (MDPs) provide a graphical and mathematical framework, which has been utilized by AI researchers to model decision theoretic planning problems. Solving MDPs has been an interesting research area for a long time, because of the slow convergence of MDP algorithms on real world domains. This paper concentrates on our advances in expediting the convergence of dynamic programming, a basic tool to solve MDPs.

Background

Markov decision processes

A Markov decision process (MDP) is a four-tuple $\langle S, A, T, C\rangle$, where S is a finite set of system states, A a finite set of actions, T the transition function or conditional probability function, and C the cost function. The MDP system develops in a sequence of discrete time slots named *stages*. At each stage t, the system is at one particular state s, where s has an associated set of applicable actions A_s^t. Applying any action makes the system change from the current state s to the next state s' and proceeds to stage $t+1$. Unlike classical AI planning, the state transition is not deterministic in MDPs. The transition function for each action a, T_a: $S \times S \to [0,1]$, tells the probabilities of state transitions under action a. $T_a(s'|s)$ stands for the probability of the system changing from s to s' by performing action a ($\sum_{s'\in S} T_a(s'|s) = 1$). The cost function $C : S \times A \to \mathbf{R}$ gives the instantaneous cost of applying an action at a state.

The *horizon* of an MDP is the total number of stages the system is evaluated. When the horizon is a finite number H, solving the MDP means finding the best action to take at each stage and state that minimizes the total expected cost. More concretely, the chosen actions $a^0, \ldots, a^{H-1}$ should minimize the expectation of the value $f(s) = \sum_{i=0}^{H-1} C(s^i, a^i)$, where $s_0 = s$. For *infinite-horizon* or *indefinite-horizon* problems, problems when the horizon is infinite or unknown, the cost is accumulated over an infinitely long path. To emphasize the relative importance of instant costs, a *discount factor* $\gamma \in [0,1]$ is used for future costs. With discount factor γ, our goal is to minimize the expectation of $f(s) = \sum_{i=0}^{\infty} \gamma^i C(s^i, a^i)$.

We consider a special type of MDPs called *goal-based MDPs* in this paper. A goal-based MDP usually has two additional components s_0 and G, where $s_0 \in S$ is the initial state and $G \subseteq S$ is a set of goal state. A solution to the MDP guides the system change from s_0 to some state in G with the smallest expected cost. A goal-based MDP is usually considered as an indefinite-horizon problem, where the horizon of the problem is finite but without an upper bound. The discount factor of a goal-based MDP is normally set to 1.

The solution of an MDP is usually represented in the form of a *policy*. Given a goal-based MDP, we define a policy $\pi : S \to A$ to be a mapping from the state space to the action space. A *value function* V_π for policy π, $V_\pi : S \to \mathbf{R}$ denotes the value of the total expected cost starting from state s and following the policy π. A policy π_1 dominates another policy π_2 if $V_{\pi_1}(s) \leq V_{\pi_2}(s)$ for all $s \in S$. An *optimal policy* π^* is a policy that is not dominated by any other policy. For goal-based MDPs, the policy and value function of different states are *stationary* (Puterman 1994). We describe the expected cost accumulated by starting at state s and following the optimal policy by the *optimal value function* V^*. Solving goal-based MDPs means to find an optimal value function and policy. Bellman (1957) showed that the expected value of a policy π can be computed using the set of value functions V^π. The value function of a policy

π is defined as:

$$V^{\pi}(s) = C(s, \pi(s)) + \gamma \sum_{s' \in S} T_{\pi(s)}(s'|s) V^{\pi}(s'), \gamma \in [0, 1]. \quad (1)$$

and the optimal value function is defined as:

$$V^{*}(s) = min_{a \in A(s)}[C(s, a) + \gamma \sum_{s' \in S} T_{a}(s'|s) V^{*}(s')], \gamma \in [0, 1]. \quad (2)$$

The *Bellman equation* is satisfied by a system of value functions in the form of Equation 1 or 2. Updating the value function of a particular state by applying the Bellman equation on that state is called a *Bellman backup*. Based on Bellman equations, we can use dynamic programming techniques to compute the exact value of value functions. An optimal policy is easily extracted by choosing an action for each state that contributes to the optimal value function.

Dynamic programming (Bellman 1957) is widely used to solve MDPs. Dynamic programming approaches explicitly store value functions of the state space, and from time to time back up states, until a time when the potential changes of value functions are very small, and we say the value functions *converge*[1]. Value iteration (Bellman 1957), for example, iteratively updates value functions by performing Bellman backups on the existing value functions. The algorithm halts when the maximum change of the value functions in the most recent iteration is smaller than a threshold value. Although value iteration converges in polynomial time (Littman, Dean, & Kaelbling 1995), its convergence is usually slow on big problems. First, it does not use initial state information to eliminate unreachable states from dynamic programming; second, backups are performed in an arbitrary order and over every state in every iteration. To overcome these problems, two types of approaches were proposed.

Previous work

The first type combines dynamic programming with *heuristic search*, so as to minimize the number of *relevant* states and the number of expansions in search. Hansen and Zilberstein (2001) proposed the first heuristic search algorithm for MDPs, named LAO*. The basic idea of LAO* is to only consider part of the state space by constructing a *partial solution graph* and searching implicitly from the initial state toward the goal state. The algorithm only expands the most promising branch of an MDP according to heuristic functions. LAO* converges much faster than VI since it only considers part of the state space. Bhuma and Goldsmith extended LAO* into BLAO* (Bhuma & Goldsmith 2003), the first bidirectional heuristic search algorithm. BLAO* searches not only in the forward direction, but also from the goal state toward the initial state in parallel. It outperformed LAO* since the heuristic values can be improved by the backward search and backups when the forward search frontier has not reached an goal state. The algorithm works the best when m_a, the maximum number of actions of each state, is large (Dai & Goldsmith 2006). LRTDP (Bonet & Geffner 2003b) and HDP (Bonet & Geffner 2003a) are two other heuristic search algorithms that use a clever *labeling* technique to mark converged states so that those states can be exempted from future search and backups.

The second type prioritizes the backups over states to decrease the number of backups, the most time-consuming portion of dynamic programming, and we call them *priority-based algorithms*. The prioritized sweeping (PS) algorithm (Moore & Atkeson 1993) was first introduced in the reinforcement learning literature, but is a general technique that has also been used in dynamic programming (Andre, Friedman, & Parr 1998; McMahan & Gordon 2005). The main consideration of PS is to order future backups more wisely by maintaining a priority queue, where the priority of each element (state) in the queue represents the potential improvement for other state values as a result of backing up that state. The priority queue is updated as the algorithm sweeps the state space. Focussed dynamic programming (FDP) (Ferguson & Stentz 2004) is another priority-based dynamic programming algorithm, where the priorities are calculated in a different way than PS. Improved Prioritized sweeping (IPS) (McMahan & Gordon 2005) is an improved version of the prioritized sweeping based dynamic programming algorithm that uses a different priority metric. It converges faster than PS and FDP.

Algorithms and results

We briefly describe our algorithms and summarize our experimental results here.

Multi-threaded BLAO*

We extended BLAO* into multi-threaded BLAO* (MBLAO*) (Dai & Goldsmith 2007a). The idea is to concurrently start several threads. One of them is the same as the forward search in BLAO*, and the rest of them are backward searches, but with different starting points. In that way we extend single-source backward search trials into multiple-source backward search trials. The reason for this change is: On the one hand, one backward search from the goal could help propagate more accurate values from the goal, but not from other sources. On the other hand, the value of a state depends on the values of all its successors, so the function of a single-source backward search is limited. This could be complemented by backward searches from other places.

Topological value iteration

Topological value iteration (TVI) (Dai & Goldsmith 2007b) is based on the observation that state values are dependent on each other. In an MDP M, if state s' is a successor state of s after applying an action a, then $V(s)$ is dependent on $V(s')$. For this reason, we want to back up s' before s. We can regard value dependency as a causal relation over the designated states. Since MDPs are cyclic, the causal relation can be cyclic and therefore quite complicated. The idea of TVI is to group states that are mutually causally related together and make them a *metastate*, and let these metastates form a new MDP M'. Then M' is no longer cyclic. In this

[1] they are sufficiently close to the optimal value functions

case, we can back up metastates in M' according to their reverse topological order. In other words, we can back up these big states in only one *virtual* iteration.

Results summary

We have done extensive experiments on the performance of MBLAO* and TVI. We summarize our results here.

We found that MBLAO* outperformed BLAO*, its single-source backward search version, and several other state-of-the-art forward heuristic search algorithms, such as LAO*, LRTDP, and HDP. The reason is that MBLAO* required the least number of backups before convergence. This result is consistent with our original considerations that backward search helps propagate more accurate heuristics from various sources. Better heuristic values not only improve the value functions, but also lead to more focused forward search. We also found that MBLAO* worked best when the initial heuristic values are not good enough (Dai 2007).

In the investigation of TVI, we found that TVI achieved the highest speedup against value iteration when the state space is evenly distributed into a number of strongly connected components. Experimental results showed that TVI converged faster, sometimes a magnitude of 10 faster, than algorithms that do not make use of the topological order of strongly connected components.

Ongoing and future work

We believe that heuristic search and priority-based approaches are very promising research topics in AI planning. We recently proposed a simple priority-based algorithm (Dai & Hansen 2007) without the use of a priority queue. Experimental results showed that it is faster than algorithms that use a priority queue. The reason is that the overhead of maintaining a priority queue sometimes exceeded its computational savings. One of our ongoing research project is on using graphical structure to expedite the convergence time in reinforcement learning MDP algorithms (Sutton & Barto 1998).

Apart from regarding the two topics individually, an integration of heuristic search and prioritization is also very interesting. For example, focussed dynamic programming (Ferguson & Stentz 2004) can be regarded as a combination of both. In the future, we plan to dig deeper along this path. We also think these two strategies can be used in combination with other common techniques such as factored MDPs, value approximation, and linear programming.

References

Andre, D.; Friedman, N.; and Parr, R. 1998. Generalized prioritized sweeping. In *Proc. of the 10th conference on Advances in neural information processing systems (NIPS-97)*, 1001–1007.

Bellman, R. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bhuma, K., and Goldsmith, J. 2003. Bidirectional LAO* algorithm. In *Proc. of Indian International Conferences on Artificial Intelligence (IICAI)*, 980–992.

Bonet, B., and Geffner, H. 2003a. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. of 18th International Joint Conf. on Artificial Intelligence (IJCAI-03)*, 1233–1238. Morgan Kaufmann.

Bonet, B., and Geffner, H. 2003b. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. 13th International Conference on Automated Planning and Scheduling (ICAPS-03)*, 12–21.

Dai, P., and Goldsmith, J. 2006. LAO*, RLAO*, or BLAO*? In *AAAI Workshop on Heuristic Search*, 59–64.

Dai, P., and Goldsmith, J. 2007a. Multi-threaded BLAO* algorithm. In *Proc. 20th International FLAIRS Conference*, 56–62.

Dai, P., and Goldsmith, J. 2007b. Topological value iteration algorithm for Markov decision processes. In *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 1860–1865.

Dai, P., and Hansen, E. A. 2007. Prioritizing Bellman backups without a priority queue. In *Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, this volumn.

Dai, P. 2007. Faster dynamic programming for Markov decision processes. Master's thesis, University of Kentucky, Lexington.

Ferguson, D., and Stentz, A. 2004. Focussed dynamic programming: Extensive comparative results. Technical Report CMU-RI-TR-04-13, Carnegie Mellon University, Pittsburgh, PA.

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

Littman, M. L.; Dean, T.; and Kaelbling, L. P. 1995. On the complexity of solving Markov decision problems. In *Proc. of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, 394–402.

McMahan, H. B., and Gordon, G. J. 2005. Fast exact planning in Markov decision processes. In *Proc. of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*.

Moore, A., and Atkeson, C. 1993. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning* 13:103–130.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley, New York.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.