A Heuristic Search Algorithm for Markov Decision Problems

Eric A. Hansen
Computer Science Department
Mississippi State University
Mississippi State, MS 39762
hansen@cs.msstate.edu

Shlomo Zilberstein
Computer Science Department
University of Massachusetts
Amherst, MA 01002
shlomo@cs.umass.edu

Abstract

LAO* is a heuristic search algorithm for Markov decision problems that is derived from the classic heuristic search algorithm AO* (Hansen & Zilberstein 1998). It shares the advantage heuristic search has over dynamic programming for simpler classes of problems: it can find optimal solutions without evaluating all problem states. In this paper, we show that the derivation of LAO* from AO* makes it possible to generalize refinements of simpler heuristic search algorithms for use in solving Markov decision problems more efficiently. We also generalize some theoretical analyses of simpler search problems to Markov decision problems.

1 Introduction

The Markov decision process (MDP), a model of sequential decision-making developed in operations research, allows reasoning about actions with uncertain outcomes in order to determine a course of action that maximizes expected utility. It has been adopted as a framework for artificial intelligence (AI) research in decision-theoretic planning (Dean *et al.* 1995) and reinforcement learning (Barto *et al.* 1995). In adopting this model, AI researchers have adopted dynamic-programming algorithms developed in operations research for solving MDPs. A drawback of dynamic programming is that it finds a solution for all problem states. In this paper, a heuristic search approach to solving MDPs is described that represents the problem space of an MDP as a search graph and uses a heuristic evaluation function to focus computation on the "relevant" problem states, that is, on states that are reachable from a given start state.

The advantage of heuristic search over dynamic programming is well-known for simpler classes of sequential decision problems. For problems for which a solution takes the form of a path from a start state to a goal state, the heuristic search algorithm A* can find an optimal path by exploring a fraction of the state space. Similarly, for problems for which a solution takes the form of a tree or an acyclic graph, the heuristic search algorithm AO* can find solutions more efficiently than dynamic programming by focusing computation on reachable states.

Neither of these heuristic search algorithms can solve the class of sequential decision problems considered in this paper. Problems of decision-theoretic planning and reinforcement learning are typically modeled as MDPs with an indefinite (or infinite) horizon. For this class of problems, the number of actions that must be taken to achieve an objective cannot be bounded, and a solution (that has a finite representation) must contain loops. Classic heuristic search algorithms, such as A* and AO*, cannot find solutions with loops. However, Hansen and Zilberstein (1998) have recently described a novel generalization of heuristic search, called LAO*, that can. They show that LAO* can solve MDPs without evaluating all problem states.

Although real-time search has been applied to MDPs before (Barto *et al.* 1995, Dean *et al.* 1995, Tash and Russell 1994, Dearden and Boutilier 1994), LAO* is the first off-line heuristic search algorithm for MDPs. This paper reviews the LAO* algorithm and describes some extensions of it that are based on its derivation from AO*. These include techniques for improving the efficiency of the algorithm, an $f = g+h$ decomposition of the evaluation function, use of a weighted heuristic, a theorem establishing a relationship between search efficiency and heuristic accuracy, a pathmax operation and a test for convergence to ϵ-optimality. These new results illustrate the relevance of the rich body of AI research on heuristic search to the problem of solving MDPs more efficiently.

2 MDPs and dynamic programming

We consider an MDP with a state set, S, and a finite action set, A. Let $A(i)$ denote the set of actions available in state i. Let $p_{ij}(a)$ denote the probability that taking action a in state i results in a transition to state j. Let $c_i(a)$ denote the expected immediate cost of taking action a in state i.

We are particularly interested in MDPs that include a start state, $s \in S$, and a set of terminal states, $T \subseteq S$. For every terminal state $i \in T$, we assume that no action can cause a transition out of it (i.e., it is an absorbing state). We also assume that the immediate cost of any action taken in a terminal state is zero. In other words, the control problem ends once a terminal state is reached. Because we are particularly interested in problems for which a terminal state is used to model a goal state, from now on we refer to terminal states as goal states.

For all non-terminal states, we assume that immediate costs are positive for every action, that is, $c_i(a) > 0$ for all states $i \notin T$ and actions $a \in A(i)$. The objective is to find a solution that minimizes the expected cumulative cost of reaching a goal state from the start state. Because the probabilistic outcomes of actions can create a non-zero probability of revisiting the same state, the worst-case number of steps needed to reach a goal state cannot be bounded. Hence, the MDP is said to have an indefinite horizon. (The results of this paper extend to MDPs without goal states, for which the objective is to optimize performance over an infinite horizon. However, we focus on MDPs with goal states because they generalize traditional AI state-space search problems.)

When an indefinite (or infinite) horizon MDP is solved using dynamic programming, a solution is represented as a mapping from states to actions, $\delta : S \rightarrow A$, called a *policy*. A policy is executed by observing the current state and taking the action prescribed for it. This representation of a solution implicitly contains both branches and loops. Branching is present because the state that stochastically results from an action determines the next action. Looping is present because the same state can be revisited under a policy.

A policy is said to be *proper* if it ensures that a goal state is reached from any state with probability 1.0. For a proper policy, the expected cumulative cost for each state i is finite and can be computed by solving the following system of $|S|$ equations in $|S|$ unknowns:

$$f^{\delta}(i) = \left[c_i(\delta(i)) + \sum_{j \in S} p_{ij}(\delta(i)) f^{\delta}(j) \right].$$

For some problems, it is reasonable to assume that all possible policies are proper. When this assumption is not reasonable, future costs can be discounted by a factor β, with $0 \leq \beta < 1$, to ensure that states have finite expected costs that can be computed by solving the following system of $|S|$ equations in $|S|$ unknowns:

$$f^{\delta}(i) = \left[c_i(\delta(i)) + \beta \sum_{j \in S} p_{ij}(\delta(i)) f^{\delta}(j) \right].$$

For problems for which neither discounting is reasonable nor all possible policies are proper, other optimality criteria – such as average cost per transition – can be adopted (Bertsekas 1995). The rest of this paper assumes discounting.

A policy δ is said to dominate a policy δ' if $f^{\delta}(i) \leq f^{\delta'}(i)$ for every state i. An optimal policy, δ^*, dominates every other policy and its evaluation function, f^*, satisfies the following system of recursive equations called the Bellman optimality equation:

$$f^*(i) = \begin{cases} 0 \text{ if } i \text{ is a goal state} \\ \text{else } min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f^*(j) \right] \end{cases}.$$

Figure 1 summarizes policy iteration, a well-known dynamic-programming algorithm for solving MDPs. Figure 2 summarizes value iteration, another dynamic-programming algorithm for solving MDPs. Both policy iteration and value iteration repeat an improvement step that updates the evaluation function for all states. Neither uses knowledge of the start state to focus computation on reachable states. As a result, both algorithms solve an MDP for all possible starting states.

1. Start with an arbitrary policy δ.
2. Repeat until the policy does not change:
   (a) Compute the evaluation function f^{δ} for policy δ by solving the following set of $|S|$ equations in $|S|$ unknowns.
   $$f^{\delta}(i) = c_i(\delta(i)) + \beta \sum_{j \in S} p_{ij}(\delta(i)) f^{\delta}(j)$$
   (b) For each state $i \in S$,
   $$\delta(i) := \arg \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f^{\delta}(j) \right]$$
   Resolve ties arbitrarily, but give preference to the currently selected action.

Figure 1: Policy iteration.

1. Start with an arbitrary evaluation function f.
2. Repeat until the error bound of the evaluation function is less than ϵ:

   For each state $i \in S$,

$$f(i) := \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f(j) \right]$$

3. Extract an ϵ-optimal policy from the evaluation function as follows. For each state $i \in S$,

$$\delta(i) := \arg \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f(j) \right]$$

Figure 2: Value iteration.

RTDP Because policy iteration and value iteration evaluate all states to find an optimal policy, they are computationally prohibitive for MDPs with large state sets. Barto, Bradtke, and Singh (1995) describe an algorithm called real-time dynamic programming (RTDP) that avoids exhaustive updates of the state set. RTDP is related to an algorithm called *asynchronous value iteration* that updates a subset of the states of an MDP each iteration. RTDP interleaves a control step with an iteration of asynchronous value iteration. In the control step, a real or simulated action is taken that causes a transition from the current state to a new state. The subset of states for which the evaluation function is updated includes the state in which the action is taken.

Figure 3 summarizes *trial-based RTDP*, which solves an MDP by organizing computation as a sequence of trials. At the beginning of each trial, the current state is set to the start set. Each control step, an action is selected based on a lookahead of depth one or more. The evaluation function is updated for a subset of states that includes the current state. A trial ends when the goal state is reached, or after a specified number of control steps.

Because trial-based RTDP only updates states that are reachable from a start state when actions are selected greedily based on the current evaluation function, states that are not reachable may not be updated. Barto, Bradtke and Singh (1995) prove that RTDP converges to an optimal solution without necessarily evaluating all problem states. They point out that this result generalizes the convergence theorem of Korf's (1990) learning real-time heuristic search algorithm (LRTA*). This suggests the following question. Given that RTDP generalizes real-time search from

1. Start with an admissible evaluation function f.
2. Perform some number of trials, where each trial sets the current state to the start state and repeats the following steps until a goal state is reached or the finite trial ends.
   (a) Find the best action for the current state based on lookahead search of depth one (or more).
   (b) Update the evaluation function for the current state (and possibly others) based on the same lookahead search used to select the best action.
   (c) Take the action selected in (2a) and set the current state to the state that results from the stochastic transition caused by the action.
3. Extract a policy from the evaluation function f as follows. For each state $i \in S$,

$$\delta(i) := \arg \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f(j) \right].$$

Figure 3: Trial-based RTDP.

problems with deterministic state transitions to MDPs with stochastic state transitions, is there a corresponding off-line search algorithm that can solve MDPs?

3 LAO*

Hansen and Zilberstein (1998) describe a heuristic search algorithm, called LAO*, that solves the same class of problems as RTDP. LAO* generalizes the classic heuristic search algorithm AO* by allowing a solution to contain loops. Unlike RTDP and other dynamic-programming algorithms, LAO* does not represent a solution (or policy) as a simple mapping from states to actions. Instead, it represents a solution as a cyclic graph with a designated start state. This representation generalizes the graphical representations of a solution used by search algorithms like A* (a simple path) and AO* (an acyclic graph). The advantage of representing a solution in the form of a graph is that it exhibits reachability among states explicitly and thus makes it easier to prune unreachable parts of the state space.

When an MDP is formalized as a graph-search problem, each state of the graph corresponds to a problem state and each arc corresponds to an action causing a transition from one state to another. When state transitions are stochastic, an arc is a *hyperarc* or *k*-

connector that connects a state to a set of k successor states, with a probability attached to each successor. A graph that contains hyperarcs is called a hypergraph and corresponds to an AND/OR graph (Martelli & Montanari 1978, Nilsson 1980). A solution to an MDP formalized as an AND/OR graph is a subgraph of the AND/OR graph called a *solution graph*, defined as follows:

- the start state belongs to a solution graph
- for every non-goal state in a solution graph, exactly one action (outgoing k-connector) is part of the solution graph and each of its successor states belongs to the solution graph
- every directed path in a solution graph terminates at a goal state

Because a heuristic search algorithm can find an optimal solution graph without evaluating all states, the entire AND/OR graph is not supplied explicitly to the search algorithm. We refer to G as the *implicit graph*; it is specified implicitly by a start state s and a successor function. The search algorithm works on an *explicit graph*, G', that initially consists only of the start state. A *tip state* of the explicit graph is a state that does not have any successors in the explicit graph. A tip state that does not correspond to a goal state is called a *non-terminal tip state*. A non-terminal tip state can be expanded by adding to the explicit graph its outgoing connectors and any successor states not already in the explicit graph.

AO*, the heuristic search algorithm that LAO* generalizes, is described by Martelli and Montanari (1973, 1978) and Nilsson (1980). It is limited to solving problems that can be represented by acyclic AND/OR graphs and the solutions it finds are acyclic. LAO* generalizes AO* to find solutions with loops in AND/OR graphs containing cycles.

Both AO* and LAO* repeatedly expand the best partial solution until a complete solution is found. The definition of a *partial solution graph* is similar to the definition of a solution graph, except that a directed path may end at a non-terminal tip state. For every non-terminal tip state i of a partial solution graph, we assume there is an admissible heuristic estimate $h(i)$ of the minimal-cost solution graph for it. A heuristic evaluation function h is said to be *admissible* if $h(i) \leq f^*(i)$ for every state i. We can recursively calculate an admissible heuristic estimate $f(i)$ of the optimal cost of any state i in the explicit graph as follows:

$$f(i) := \begin{cases} 0 \text{ if } i \text{ is a goal state} \\ h(i) \text{ if } i \text{ is a non-terminal tip state} \\ \text{else } min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f(j) \right]. \end{cases}$$

For a partial solution graph that does not contain loops, these heuristic estimates can be calculated by dynamic programming in the form of a simple backwards induction algorithm, and this is what AO* does. For a partial solution graph that contains loops, dynamic programming cannot be performed in the same way. However, it can be performed by using policy iteration or value iteration. This simple generalization of AO* creates the algorithm LAO*. The shared structure of AO* and LAO* is summarized in Figure 4.

In the simplest version of both AO* and LAO*, dynamic programming is performed on the set of states that includes the expanded state and all of its ancestors in the explicit graph (Martelli & Montanari 1973). Martelli and Montanari (1978) and Nilsson (1980) note that only ancestor states that can reach the expanded state along marked actions (i.e., by choosing the best action for each state) can have their costs changed in the dynamic-programming step of AO*. This also applies to LAO* and limits the set of states on which the dynamic-programming step of LAO* is performed. Note that when dynamic programming is performed on a subset of the states in the explicit graph, the costs of states that are not in the subset are treated as constants by the algorithm because they cannot be affected by any change in the cost of the expanded state or its ancestors.

The dynamic-programming step of LAO* can be performed using either policy iteration or value iteration. An advantage of using policy iteration is that it computes an exact cost for each state of the explicit graph after a finite number of iterations, based on the heuristic estimates at the tip states. When value iteration is used in the dynamic-programming step, convergence to exact state costs is asymptotic. However, this disadvantage may be offset by the improved efficiency of value iteration for larger problems.

Descriptions of the AO* algorithm usually include a solve-labeling procedure. Briefly, a state is labeled solved if it is a goal state or if all of its successor states are labeled solved. We have omitted the solve-labeling procedure from our description of AO* and LAO* to make the essential algorithm clearer.

In both AO* and LAO*, the fringe of the best partial solution graph may contain many unexpanded states and the choice of which to expand next is nondeterministic. That is, both AO* and LAO* work correctly no matter what heuristic is used to select which non-

1. The explicit graph G' initially consists of the start state s.

2. While the best solution graph has some non-terminal tip states:

   (a) *Expand best partial solution*: Expand some non-terminal tip state n of the best partial solution graph and add any new successor states to G'. For each new state i added to G' by expanding n, if i is a goal state then $f(i) := 0$; else $f(i) := h(i)$.

   (b) *Update state costs and mark best actions*:

      i. Create a set Z that contains the expanded state and all of its ancestors in the explicit graph along marked action arcs. (I.e., only include ancestor states from which the expanded state can be reached by following the current best solution.)

      ii. Perform dynamic programming on the states in set Z to update state costs and determine the best action for each state. (AO* performs dynamic programming using backwards induction. LAO* uses policy iteration or value iteration.)

3. If this is AO* or LAO* using policy iteration, return the solution graph.

   Else if this is LAO* using value iteration, perform value iteration on the states in the best partial solution until one of the following two conditions is met. (i) If the error bound falls below ϵ, exit with an ϵ-optimal solution. (ii) If the best partial solution changes so that it has an unexpanded tip state, go to step 2.

Figure 4: AO* and LAO*.

terminal tip state of the best partial solution graph to expand next. A well-chosen state selection heuristic can improve performance, however. Possibilities include expanding the state with the highest probability of being reached from the start state or expanding the state with the least cost.

LAO* bears a close similarity to an "envelope" approach to policy and value iteration described by Dean *et al.* (1995) and Tash and Russell (1994). However, the envelope approach is not derived from AO* and no proof is given that it converges to an optimal solution without evaluating all problem states.

Convergence test Both AO* and LAO* converge when the best solution graph does not have any non-terminal tip states. When policy iteration is used to compute exact state costs in the dynamic-programming step, the solution to which LAO* converges is optimal. When state costs are updated approximately using value iteration in the dynamic-programming step, the solution to which LAO* converges may not be optimal. In this case, an additional convergence test can be used to determine whether the solution is ϵ-optimal, for any ϵ. The convergence test is summarized in step 3 of Figure 4.

The test is simple and consists of performing value iteration on the set of states visited by the current best solution graph. This set of states may change from one iteration of value iteration to the next because the best action for a state can be changed by value iteration. If at any point the best solution graph changes so that it includes a non-terminal tip state, control is passed from this convergence test back to the main algorithm so that the current solution can be expanded and re-evaluated. Otherwise, the Bellman residual is computed each iteration and used to determine the error bound of the solution. Because the estimated cost of every state in the explicit graph is admissible, the Bellman residual only needs to be computed for the states visited by the best solution to determine the error bound of the solution – as long as the best solution is a complete solution. Value iteration is repeated until the error bound of the solution falls below ϵ. When it does, LAO* terminates with an ϵ-optimal solution.

Admissibility It is possible to show that LAO* shares the properties of AO* and other heuristic search algorithms. Given an admissible heuristic evaluation function, all state costs in the explicit graph are admissible after each step and LAO* converges to an optimal or ϵ-optimal solution without (necessarily) evaluating all problem states. The following proofs are adapted from (Hansen & Zilberstein 1998).

Theorem 1 *If the heuristic evaluation function h is admissible and policy iteration is used to perform the dynamic programming step of LAO*, then:*

1. *$f(i) \leq f^*(i)$ for every state i, after each step of LAO**
2. *$f(i) = f^*(i)$ for every state i of the best solution graph, when LAO* terminates*
3. *LAO* terminates after a finite number of iterations*

Proof: (1) The proof is by induction. Every state $i \in G$ is assigned an initial heuristic cost estimate and $h(i) \leq f^*(i)$ by the admissibility of the heuristic evaluation function. The forward search step expands

the best partial solution graph and does not change the cost of any states and so it is sufficient to consider the dynamic programming step. We make the inductive assumption that at the beginning of this step, $f(i) \leq f^*(i)$ for every state $i \in G$. If all the tip states of G' have optimal costs, then all the nontip states in G' must converge to their optimal costs when policy iteration is performed on them by the convergence proof for policy iteration. But by the induction hypothesis, all the tip states of G' have admissible costs. It follows that the nontip states in G' must converge to costs that are as good or better than optimal when policy iteration is performed on them only.

(2) The search algorithm terminates when the best solution graph for s is complete, that is, has no unexpanded states. For every state i in this solution graph, it is contradictory to suppose $f(i) < f^*(i)$ since that implies a complete solution that is better than optimal. By (1) we know that $f(i) \leq f^*(i)$ for every state in G'. Therefore $f(i) = f^*(i)$.

(3) It is obvious that LAO* terminates after a finite number of iterations if the implicit graph G is finite, or equivalently, the number of states in the MDP is finite. (When the state set is not finite, it may still converge in some cases.) □

Theorem 2 *If the heuristic evaluation function h is admissible and value iteration is used to perform the dynamic programming step of LAO*, then:*

1. *$f(i) \leq f^*(i)$ for every state i at every point in the algorithm*
2. *$f(i)$ converges to within ϵ of $f^*(i)$ for every state i of the best solution graph, after a finite number of iterations*

Proof: (1) The proof is by induction. Every state $i \in G$ is assigned an initial heuristic cost estimate and $f(i) = h(i) \leq f^*(i)$ by the admissibility of the heuristic evaluation function. We make the inductive hypothesis that at some point in the algorithm, $f(i) \leq f^*(i)$ for every state $i \in G$. If a value iteration update is performed for any state i,

$$\begin{aligned} f(i) &= \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f(j) \right] \\ &\leq \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f^*(j) \right] = f^*(i), \end{aligned}$$

where the last equality restates the Bellman optimality equation.

(2) Because the graph is finite, LAO* must eventually find a complete solution graph. Because an iteration of value iteration reduces the error bound by an amount proportional to β, the error bound of the solution can be made arbitrarily small after a finite number of iterations. □

4 Extensions

LAO* can solve the same class of MDPs as RTDP and both algorithms converge to an optimal solution without evaluating all problem states. The derivation of LAO* from the heuristic search algorithm AO* has an important advantage, however; it makes it easier to generalize refinements of simpler heuristic search algorithms for use in solving MDPs more efficiently. To illustrate this, we describe some extensions of LAO* that are suggested by its derivation from AO* (and ultimately, from A*).

4.1 $f = g + h$

The A* search algorithm relies on a familiar $f = g + h$ decomposition of the evaluation function. Chakrabarti *et al.* (1988) have shown that a similar decomposition of the evaluation function is possible for AO* and that it supports a weighted heuristic version of AO*. We first show that this $f = g + h$ decomposition of the evaluation function can be extended to LAO*. Then we describe a similar weighted heuristic version of LAO*.

Each time a state i is generated by A*, a new estimate $f(i)$ of the optimal cost from the start state to a goal state is computed by adding $g(i)$, the cost-to-arrive from the start state to i, to $h(i)$, the estimated cost-to-go from i to a goal state. Each time a state i is generated by AO* or LAO*, the optimal cost from the start state to a goal state is also re-estimated. In this case, it is re-estimated by updating the estimated cost of every ancestor state of i, including the start state. For the start state (and similarly for any other ancestor state), the estimated cost-to-go, denoted $f(s)$, is decomposed into $g(s)$, the cost-to-arrive from the start state to a state on the fringe of the best partial solution, and $h(s)$, the estimated cost-to-go from the fringe of the best partial solution to a goal state. In other words, $g(s)$ represents the part of the solution cost that has been explicitly computed so far and $h(s)$ represents the part of the solution cost that is still only estimated.

Note that after a state i is generated by AO* or LAO*, and before it is expanded, $g(i)$ equals zero and $f(i)$ equals $h(i)$. After state i is expanded, $f(i)$ is updated in the same manner as before:

$$f(i) := \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f(j) \right],$$

where the discount factor β is used with discounted MDPs only. Having determined the action a that optimizes $f(i)$, the algorithm proceeds to update $g(i)$ and $h(i)$ as follows:

$$g(i) := c_i(a) + \beta \sum_{j \in S} p_{ij}(a) g(j),$$
$$h(i) := \beta \sum_{j \in S} p_{ij}(a) h(j).$$

Note that once LAO* converges to a complete solution, $h(i) = 0$ and $f(i) = g(i)$ for every state i in the solution.

The possibility of decomposing the evaluation function of an MDP in this way elegantly reflects the interpretation of an MDP as a heuristic search problem. It also makes it possible to create a weighted heuristic that can improve the efficiency of LAO* in exchange for a bounded decrease in the optimality of the solution it finds.

4.2 Weighted heuristic

Pohl (1973) first described how to create a weighted heuristic and showed that it can improve the efficiency of A* in exchange for a bounded decrease in solution quality. Given the familiar decomposition of the evaluation function, $f(i) = g(i) + h(i)$, a weight w, with $0.5 \leq w \leq 1$, is used to create a weighted heuristic, $f(i) = (1 - w)g(i) + wh(i)$. Use of this heuristic guarantees that solutions found by A* are no more than a factor of $w/(1 - w)$ worse than optimal (Davis *et al.*, 1989). Given a similar $f = g + h$ decomposition of the evaluation function computed by LAO*, it is straightforward to create a weighted version of LAO* that can find an ϵ-optimal solution by evaluating a fraction of the states that LAO* might have to evaluate to find an optimal solution.

We also note that a $f = g + h$ decomposition of the evaluation function and a weighted heuristic can be used with RTDP. In fact, the idea of weighting a heuristic in order to find a bounded-optimal solution quickly is also explored by Ishida and Shimbo (1996) for Korf's LRTA* algorithm, which can be viewed as a special case of RTDP for solving deterministic search problems. They create a weighted heuristic by simply multiplying the initial heuristic estimate of each state's value by a weight. (This approach is less flexible than one that relies on the $f = g+h$ decomposition because it does not allow the weight to be adjusted.) They find that a weighted heuristic can significantly reduce the number of states explored (expanded) by LRTA* without significantly decreasing the quality of the solution it finds.

4.3 Heuristic accuracy and search efficiency

In all heuristic search algorithms, three sets of states can be distinguished. The implicit graph contains all problem states. The explicit graph contains those states that are evaluated in the course of the search. The solution graph contains those states that are reachable from the start state when the best solution is followed. The objective of a best-first heuristic search algorithm is to find an optimal solution (for a given start state) while generating as small an explicit graph as possible.

Like all heuristic search algorithms, the efficiency of LAO* depends crucially on the heuristic evaluation function that guides the search. The more accurate the heuristic, the fewer states need to be evaluated to find an optimal solution, that is, the smaller the explicit graph generated by the search algorithm. For A*, the relationship between heuristic accuracy and search efficiency has been made precise. Given two heuristic functions, h_1 and h_2, such that $h_1(i) \leq h_2(i) \leq f^*(i)$ for all states i, the set of states expanded by A* when guided by h_2 is a subset of the set of states expanded by A* when guided by h_1 (Nilsson 1980).

A result this strong does not hold for AO*, or by extension, for LAO*. The reason it does not is that the selection of the next state to expand on the fringe of the best partial solution is nondeterministic. Although AO* and LAO* work correctly no matter what state on the fringe of the best partial solution is expanded next, a particular choice may result in some states being expanded that would not be if the choice were different. Nevertheless, Chakrabarti *et al.* (1988) show that a weaker result does hold for AO* and it is straightforward to extend this result to LAO*.

Chakrabarti *et al.* (1998) consider the worst-case set of states expanded by a search algorithm. Let V denote the worst-case set of states expanded by LAO*, defined as follows:

- the start state s is in V
- a state i is in V if there exists a partial solution graph p such that:
  - $f^p(s) \leq f^*(s)$
  - every non-tip state of p is in V
  - i is a non-terminal tip state of p
- no other states are in V

Given this definition, we have the following theorem.

Theorem 3 *Given two heuristic functions, h_1 and h_2, such that $h_1(i) \leq h_2(i) \leq f^*(i)$ for all states i,*

the worst-case set of states expanded by LAO when guided by h_2 is a subset of the worst-case set of states expanded by LAO* when guided by h_1.*

Proof: For any partial solution graph for start state s, we have

$$\begin{aligned} f_1(s) &= g(s) + h_1(s), \\ f_2(s) &= g(s) + h_2(s). \end{aligned}$$

Since $h_1(s) \leq h_2(s)$, we have $f_1(s) \leq f_2(s)$. Thus, if $f_2(s) \leq f^*(s)$, we also have $f_1(s) \leq f^*(s)$. It follows that the worst-case set of states expanded by LAO* when guided by h_2 must be a subset of the worst-case set of states expanded by LAO* when guided by h_1. □

In other words, although a more accurate heuristic does not necessarily make LAO* more efficient, it makes it more efficient in the worst case. This result only holds for a "pure" version of LAO* that updates state costs exactly in the dynamic-programming step. If LAO* updates state costs approximately, for example, by using value iteration with a relaxed criterion for convergence, the effect on the set of states expanded in the worst case is not clear.

4.4 Pathmax

In A* search, a heuristic evaluation function h is said to be *consistent* if it is both admissible and $h(i) \leq c_i(a) + h(j)$ for every state i and action a, where j is the successor state. Consistency is a desirable property because it ensures that state costs increase monotonically as the algorithm converges. If a heuristic evaluation function is admissible but not consistent, state costs need not increase monotonically. However, they can be made to do so by adding a *pathmax* operation to the update step of the algorithm. This common improvement of A* and AO* can also be made to LAO*.

For LAO*, a heuristic evaluation function h is said to be *consistent* if it is both admissible and $h(i) \leq c_i(a) + \beta \sum_j p_{ij}(a)h(j)$ for every state i and action $a \in A(i)$. A pathmax operation is added to LAO* by changing the formula used to update state costs as follows:

$$f(i) := \max \left[f(i), \min_{a \in A(i)} \left[c_i(a) + \beta \sum_{j \in S} p_{ij}(a) f(j) \right] \right].$$

As with the classic heuristic search algorithms A* and AO*, use of the pathmax operation in LAO* can reduce the number of states that are expanded to find an optimal solution, when the heuristic is not consistent.

5 Performance

We have implemented LAO* and tested it on a suite of problems that includes the gridworld and racetrack problems used to test the performance of RTDP and similar envelope algorithms (Barto *et al.* 1995, Dean *et al.* 1995, Tash & Russell 1994). The largest of these test problems is a racetrack problem with a little over 22,000 states, used as a test problem by Barto *et al.* (1995). Problems of this size are too small for a comprehensive evaluation of LAO*, but they do allow us to observe some of the properties of the algorithm. For example, a simple admissible heuristic for the racetrack problem sets the initial heuristic cost of each state to zero. In this case, LAO* finds an optimal solution that visits 2,107 states after building an explicit graph that contains 15,824 states. A better admissible heuristic for this problem is computed by beginning from the goal state and determining, using Dijkstra's algorithm, the shortest number of actions it can take to reach the goal for each state. Using this heuristic, LAO* finds the same optimal solution after expanding only 11,127 states. Weighting this heuristic with a weight of 0.7, LAO* found a solution within one decimal point of the optimal expected cost after expanding only 9,832 states. These results illustrate the importance of heuristic accuracy in limiting search complexity.

In our experiments, we found that a naive implementation of LAO* is not necessarily as efficient as RTDP, or even value iteration. Expanding just one state at a time, for example, can slow LAO* by causing excessive iterations of the dynamic-programming step. The performance of LAO* can usually be improved – often dramatically – by expanding multiple states on the fringe of the best partial solution before updating the costs of ancestor states using dynamic progamming.

Updating all ancestor states of an expanded state in the dynamic-programming step can also slow the algorithm unnecessarily. For some problems, surprisingly, an expanded state can have many more ancestors than there are states in the best partial solution. (This is true for the racetrack problem, for example, because hitting a wall causes a transition back to the start state. This makes almost every state of the problem an ancestor of the start state, and thus an ancestor of any expanded state – even though a fraction of these states may be reachable from the start state by following the best solution.) In such cases, performing dynamic programming only on ancestor states of an expanded state that are also part of the current best solution can significantly reduce the number of states on which dynamic programming is performed, and does so without affecting the convergence of the algorithm. Techniques

like *prioritized sweeping* (Moore & Atkinson 1993) may also be used to accelerate the value-iteration step of LAO* by focusing updates where they will have the most impact. We do not discuss these possibilities further in this paper. For most of the simple problems we tested, the performance of a careful implementation of LAO* – in particular, one that expands multiple states at a time on the fringe of the best partial solution – is competitive with RTDP.

We have also used LAO* to solve some partially observable MDPs (POMDPs). A POMDP is a generalization of an MDP in which the state of the process is not directly observed; instead, an observation is recieved that is probabilistically related to the underlying state (Kaelbling, Littman & Cassandra 1998). A POMDP can be formalized as a completely observable MDP for which the state set is the set of all possible probability distributions – called belief states – over the underlying states. Although the set of all possible belief states is uncountably infinite, there is a subset of POMDPs for which an optimal policy visits a finite number of belief states, starting from a particular belief state. Simple examples include the tiger-behind-the-door problem of Kaelbling *et al.* (1998) and the maze problem of McCallum (1993). Inspection/replacement problems, a widely-studied class of problems in operations research, also have this property (Thomas *et al.* 1991). Taking either a replace or an inspect action (assuming perfect inspections) creates a loop to some previously visited belief state and a policy that visits a finite number of belief states. A similar class of problems that can be solved using LAO* are MDPs with delayed information (Brooks & Leondes 1973). We have used LAO* to find optimal solutions to small examples of all of these types of problems.

The assumption that an optimal solution visits a finite number of belief states severely limits the range of POMDPs for which LAO* is an effective approach, and we do not propose LAO* as an approach to POMDPs in general. Nevertheless, the fact that LAO* can find an optimal solution for some POMDPs by evaluating a search graph containing a small number of belief states is notewothy. It vividly illustrates the ability of heuristic search to converge to an optimal solution without evaluating all possible states – in this case, an uncountably infinite number of possible belief states.

6 Discussion

This paper describes an enhanced version of the LAO* algorithm first described by Hansen and Zilberstein (1998). The enhancements introduced include a rule for limiting the set of ancestor states on which the dynamic-programming step is performed, a test for convergence to ϵ-optimality when value iteration is used in the dynamic-programming step, an $f = g + h$ decomposition of the evaluation function, use of a weighted heuristic, a theorem establishing a relationship between search efficiency and heuristic accuracy and a pathmax operation. We have also reported some observations of the algorithm's performance.

Real-time search has been used before to solve MDPs (Barto *et al.* 1995, Dean *et al.* 1995, Tash & Russell, 1994, Dearden & Boutilier 1994). LAO* is unique in that it is an *off-line* heuristic search algorithm for MDPs. (Its representation of a solution as a cyclic graph also distinguishes it from earlier work on using search to solve MDPs.) We have stressed the derivation of LAO* from the classic heuristic search algorithm AO* because we believe this derivation provides a foundation for a heuristic search approach to MDPs. It also makes it easier to generalize enhancements of classic search algorithms for use in solving MDPs more efficiently, as this paper illustrates. Most of the enhancements described in this paper generalize similar enhancements of AO*. The enhancements of LAO* we have described can also be used to improve the efficiency of RTDP. Finally, this work provides a theoretical foundation for the closely-related envelope algorithms of Dean *et al.* (1995) and Tash and Russell (1994).

Like RTDP, LAO* is especially useful for problems for which an optimal solution, given a start state, visits a fraction of the state space. It is easy to give examples of MDPs like this. But there are also MDPs with optimal solutions that visit the entire state space. An interesting question is how to recognize problems for which an optimal solution visits a fraction of the state space and distinguish them from problems for which an optimal solution visits most, or all, of the state space. Even for problems for which an optimal solution visits the entire state space, LAO* may find useful partial solutions by focusing computation on states that are most likely to be visited from a start state – a possibility that motivates the envelope algorithms of Dean *et al.* (1995) and Tash and Russell (1994).

Acknowledgments

Support for this work was provided in part by the National Science Foundation under grants IRI-9624992 and IRI-9634938.

References

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

Bertsekas, D. (1995) *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA.

Brooks, D.M. and Leondes, C.T. (1993) Markov decision processes with state-information lag. *Operations Research* pp. 904–907.

Chakrabarti, P.P.; Ghose, S.; & DeSarkar, S.C. (1988) Admissibility of AO* when heuristics overestimate. *Artificial Intelligence* 34:97-113.

Davis, H.W.; Bramanti-Gregor, A.; and Wang, J. (1989) The advantages of using depth and breadth components in heuristic search. In Z.W. Ras and L. Saitta, eds., *Methodologies for Intelligent Systems*, Vol. 3, North-Holland, Amsterdam, The Netherlands, pp. 19–28.

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

Dearden, R and Boutilier, C. (1994) Integrating planning and execution in stochastic domains. In Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, pp. 162–169. Washington, D.C.

Hansen, E.A. and Zilberstein, S. (1998) Heuristic search in cyclic AND/OR graphs. Proceedings of the Fifteenth National Conference on Artificial Intelligence, 412–418. Madison, WI.

Ishida, T. and Shimbo, M. (1996) Improving the learning efficiencies of realtime search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 305–310. Portland, OR.

Kaelbling, L.P; Littman, M.L; and Cassandra, A.R. (1998) Planning and Acting in partially observable stochastic domains. em Artificial Intelligence 101:99–134.

Korf, R. (1990) Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Martelli, A. and Montanari, U. (1973) Additive AND/OR graphs. In Proceedings of the Third International Joint Conference on Artificial Intelligence, 1–11. Stanford, CA.

Martelli, A. and Montanari, U. (1978) Optimizing decision trees through heuristically guided search. *Communications of the ACM* 21(12):1025–1039.

McCallum, R.A. (1993) Overcoming incomplete perception with utile distinction memory. In Proceedings of the Tenth International Machine Learning Conference, pp. 190–196. Amherst, MA.

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

Nilsson, N.J. (1980) *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga Publishing Company.

Pohl, I. (1973) First results on the effect of error in heuristic search. *Machine Intelligence* 5:219–236.

Tash, J. and Russell, S. (1994) Control strategies for a stochastic planner. In Proceedings of the Twelth National Conference on Artificial Intelligence, 1079–1085. Seattle, WA.

Thomas, L.C.; Gaver, D.P.; and Jacobs, P.A. (1991) Inspection models and their application. *IMA Journal of Mathematics Applied in Business and Industry* 3:383–303.