Iterative Bounding LAO*

Håkan Warnquist[1,2] and **Jonas Kvarnström**[2] and **Patrick Doherty**[2]

Abstract. Iterative Bounding LAO* is a new algorithm for ϵ-optimal probabilistic planning problems where an absorbing goal state should be reached at a minimum expected cost from a given initial state. The algorithm is based on the LAO* algorithm for finding optimal solutions in cyclic AND/OR graphs. The new algorithm uses two heuristics, one upper bound and one lower bound of the optimal cost. The search is guided by the lower bound as in LAO*, while the upper bound is used to prune search branches. The algorithm has a new mechanism for expanding search nodes, and while maintaining the error bounds, it may use weighted heuristics to reduce the size of the explored search space. In empirical tests on benchmark problems, Iterative Bounding LAO* expands fewer search nodes compared to state of the art RTDP variants that also use two-sided bounds.

1 INTRODUCTION

In this paper, we study probabilistic planning problems formulated as stochastic shortest path problems. In these problems an absorbing goal state should be reached from a single initial state with a minimal expected cost. Each action has positive cost and may have a finite number of probabilistic outcomes. Some examples of such problems are robot control with noisy actuators and troubleshooting of technical equipment where an unknown fault should be isolated and repaired.

A stochastic shortest path problem can be modeled as a general search problem in AND/OR graphs and solved with algorithms such as LAO* [6] or Real-Time Dynamic Programming (RTDP) [1]. The output of these algorithms is a policy that maps states to actions.

We present a new algorithm for solving stochastic shortest path problems. This algorithm, Iterative Bounding LAO*, is based on LAO* and uses two heuristics, one upper bound and one lower bound of the minimal expected cost. The lower bound is used to guide search as in LAO* while the upper bound is used to prune search branches. Iterative Bounding LAO* is an online ϵ-optimal probabilistic planning algorithm that can output policies with proven bounds at any time. These bounds are reduced iteratively over time. The algorithm has a new mechanism for expanding search nodes, and while maintaining the error bound, it may use weighted heuristics to reduce the search space.

Two-sided bounds have been used in other algorithms based on RTDP, such as BRTDP [8], FRTDP [12], and VPI-RTDP [11]. In empirical tests on benchmark problems Iterative Bounding LAO* is shown to be competitive with these state-of-the-art RTDP variants and expands fewer nodes.

[1] Affiliated with Scania CV AB.
[2] Dept. of Computer and Information Science,Linköping University, email: {g-hakwa,jonkv,patdo}@ida.liu.se

2 PROBLEM FORMULATION

As stated in the introduction, the probabilistic planning problems considered in this paper have an absorbing goal state that should be reached at a minimum expected cost from a given initial state. Such a problem corresponds to a stochastic shortest path problem that is described by a set of states S, a finite set of actions A, a cost function $q : A \times S \mapsto \mathbb{R}_+$, a transition function $P : A \times S^2 \mapsto [0, 1]$, an initial state s_0 and a set of absorbing goal states S_g. When a is performed in a state s, another state s' is reached with the probability $P(s'|a, s)$. The set of successor states $succ(s, a)$ consists of all states s' where $P(s'|a, s) > 0$. Since the goal states are absorbing, for all actions a and goal states s, $q(a, s) = 0$ and $P(s|a, s) = 1$. For all other states $q(a, s) > 0$. A policy π is a function that maps states to actions. For any action a, let T_a be an operator on functions $f : S \mapsto \mathbb{R}$ such that for any state s,

$$T_a f(s) = q(a, s) + \sum_{s' \in succ(s,a)} P(s'|a, s) f(s').$$

Definition 1 (Value function)**.** Given a policy π and a state s, the *value function* $V_\pi(s)$ returns the expected cost of reaching a goal state from s:

$$V_\pi(s) = \begin{cases} 0 & \text{if } s \text{ is a goal state,} \\ T_{\pi(s)} V_\pi(s) & \text{otherwise.} \end{cases}$$

Definition 2 (Optimal policy)**.** An optimal policy π^* is a policy such that for all states s,

$$\pi^*(s) = \arg\min_a T_a V_{\pi^*}(s)$$

Finding $\pi^*(s)$ for a given state s is equivalent to solving the Bellman optimality equation:

$$V_{\pi^*}(s) = \begin{cases} 0 & \text{if } s \text{ is a goal state,} \\ \min_a T_a V_{\pi^*}(s) & \text{otherwise.} \end{cases} \quad (1)$$

Definition 3 (ϵ-optimal policy)**.** Given an $\epsilon > 0$, a policy π is ϵ-optimal if $V_\pi(s_0) \leq (1 + \epsilon) V_{\pi^*}(s_0)$.

The problem is to find an ϵ-optimal policy for a given value of ϵ.

3 RELATED WORK

Real time dynamic programming (RTDP) [1] is an efficient algorithm for solving stochastic shortest path problems in an on-line setting where good anytime properties are important. RTDP solves (1) for the subset of the state space that is reachable from the initial state by performing a series of asynchronous value iteration back-ups. In such a back-up, an estimate of V_{π^*} is computed from previous estimates of V_{π^*} by evaluating (1). If every state is backed up infinitely many times, the estimates will converge to V_{π^*} regardless of which value they are initialized with. The states that are backed up are chosen through random depth-first trials starting from the initial state. A

drawback of this method is that due to the random exploration of the state space many states will be backed up even if their values have already converged.

In BRTDP [8], FRTDP [12], and VPI-RTDP [11], all of which are extensions to RTDP, an upper bound of the optimal value function is used to help decide if a state has converged or not. In both BRTDP and FRTDP states with high difference in lower and upper bounds are given priority in the RTDP trials. In BRTDP the trials are random while in FRTDP they are deterministic. The algorithm VPI-RTDP uses a slightly different approach. Here, successor states are chosen based on an estimate of the expected improvement in decision quality when updating the state's value.

LAO* [6] is another algorithm that can solve the stochastic shortest path problem. This algorithm does not use deep trials like RTDP. Instead it uses a heuristic to expand states in a best-first manner. It is similar to the AND/OR graph search algorithm AO* [9] and the deterministic search algorithm A*. The main drawback of LAO* is its relatively expensive dynamic programming step between expansions which is explained in further detail below. Therefore, an improved variant of LAO*, ILAO*, is proposed in [6] where states are backed up at most once.

4 ALGORITHM

Iterative Bounding LAO* is derived from LAO* [6]. We will therefore begin with an overview of the standard LAO* algorithm.

4.1 Overview of LAO*

LAO* extends AO* [9], a search algorithm for acyclic AND/OR graphs, to find solutions in cyclic graphs and may use many of the enhancements developed for A* such as weighted heuristics.

LAO* searches an AND/OR graph, which can be represented as a directed hypergraph. We can create such a graph G for our problem as follows. Let the nodes in the graph be states and let s_0 belong to G. For every action a applicable in a state $s \in G \setminus S_g$, let the states in $succ(s, a)$ also belong to G and add one outgoing hyperedge leading from s to the states in $succ(s, a)$. This results in a graph where all leaves are goal states.

A *solution* to a search graph G is a subgraph $G_\pi \subseteq G$ satisfying the following constraints. First, the initial state s_0 is part of G_π. Second, only states that are leaves in G can be leaves in G_π. Third, for any non-leaf s in G_π, there is exactly one outgoing hyperedge corresponding to a chosen action a to be performed in that state, and all possible successor states $succ(s, a)$ of that action belong to G_π.

Given a solution graph G_π, we can directly generate a policy π where for all $s \in G_\pi \setminus S_g$, $\pi(s)$ is defined by the single outgoing hyperedge from s. Such a policy is *complete*, in the sense that it specifies an action for every non-goal state that is reachable by following the policy within G.

Let G' be an arbitrary subgraph of G containing the initial state s_0. Further, let G'_π be a solution to this subgraph where each non-leaf $s \in G'$ has an outgoing edge labeled with an action

$$\pi(s) = \arg\min_{a \in A} T_a f(s). \tag{2}$$

If all leaves in G'_π are goal states, then G'_π must also be a solution to G and therefore corresponds to a complete policy π for G. Since the subgraph is arbitrary, there may also be leaves that are not goal states. In this case, G'_π can be said to correspond to a partial policy π for G, which can lead to non-goal states for which no action is specified. LAO* can expand such a partial policy by specifying actions for non-goal leaves, thereby incrementally expanding G' until its solution is also a solution to G without necessarily exploring all of G.

A state s in a solution G'_π is evaluated with the evaluation function

$$f(s) = \begin{cases} h(s) & \text{if } s \text{ is a leaf state in } G', \\ T_{\pi(s)} f(s) & \text{otherwise,} \end{cases} \tag{3}$$

where $h(s)$ is a heuristic estimate of the optimal expected cost such that $0 \leq h(s) \leq V_{\pi^*}(s)$. If π is a complete policy then $f(s) = V_\pi(s)$ since in each leaf, $h(s) = V_\pi(s) = 0$. It is possible that a complete policy have states from which a goal state is unreachable. However, the expected cost of such a policy is infinite.

The LAO* algorithm is shown in Figure 1. LAO* is initialized with an explicit search graph $G' \subseteq G$ consisting only of the initial state s_0. The set $fringe(G'_\pi)$ consists of all non-goal leaf states in a solution G'_π reachable from s_0 The intention is to ensure eventually that $fringe(G'_\pi) = \emptyset$, i.e. that there is an action to perform for every non-goal state. Until this is the case, one or more states s in $fringe(G'_\pi)$ are expanded and, for each action a, the successors $succ(s, a)$ are added to G'.

After the expansion step, (3) is evaluated for all ancestors of the newly expanded states. This may be done using either policy iteration or value iteration, but it is shown in [6] that LAO* is faster with value iteration. During value iteration, for each state s an action satisfying (2) is marked and a new value for $f(s)$ is calculated from the previous f-values of all other states. Each such update is called a back-up and this is done until the f-values converge over all states. When $fringe(G'_\pi) = \emptyset$, LAO* performs value iteration on all states in G'_π until either the f-values converge or some non-goal state appears among the leaf states of G'_π in which case LAO* goes back to step 2. When all leaves in G'_π are goal states and the f-values have properly converged, $fringe(G'_\pi) = \emptyset$ and $\pi = \pi^*$.

1. $G' \leftarrow \{s_0\}$
2. **while** $fringe(G'_\pi) \neq \emptyset$ **do**
   (a) Expand one or more states in $fringe(G'_\pi)$ and add any successor states to G'.
   (b) Perform value iteration on all ancestor states of the newly expanded states.
3. Perform value iteration on all states in G'_π.
   if $fringe(G'_\pi) \neq \emptyset$ **then** go to step 2.
4. **return** π

Figure 1: LAO*.

4.2 Iterative Bounding LAO*

The new algorithm is based on LAO*. It maintains two-sided bounds on the optimal solution cost and uses these to prune search branches when the error bound on the optimal solution cost is below a certain threshold. To perform well in an on-line setting this threshold is dynamically changed, starting off with a high value that is successively reduced as better solutions are found. The most recent bounds on the optimal solution cost are always available and the user may use this information to decide when to stop the search.

The Iterative Bounding LAO* algorithm is shown in Figure 2. Throughout this algorithm, whenever a state s is visited for the first time a lower bound f_l and an upper bound f_u are calculated such that $f_l(s) \leq V_{\pi^*}(s) \leq f_u(s)$. The computation of these bounds is described in Section 4.3.

In step 1, an initial search graph G' is created, consisting only of the initial state s_0. The outer loop in step 2 continues indefinitely until stopped by the user. In step 2a the error threshold $\bar{\epsilon}$ is initialized to be a factor $\alpha < 1$ times the current error bound $\hat{\epsilon}(s_0)$ in the initial state. The computation of the error bound is described in Section 4.4.

The inner loop in step 2b is similar to the LAO* algorithm where fringe states are expanded until a partial policy is found such that the initial state is solved within the current required bound, i.e. $\hat{\epsilon}(s_0) \leq \bar{\epsilon}$. The set $fringe(G'_{\pi_l})$ consists of all leaf states in G'_{π_l}, the partial solution graph of the lower bound policy, that have $\hat{\epsilon}(s) > \bar{\epsilon}$ and consequently are not yet solved within the current error bound. If $fringe(G'_{\pi_l}) \neq \emptyset$, we select a subset S_{expand} of $fringe(G'_{\pi_l})$ that is expanded as described in Section 4.5. When a state is expanded, all successors to that state are inserted in G' and the lower and upper bounds for the successor states are calculated.

After the expansions, all ancestors of the newly expanded states, $ancestors(S_{expand})$, are backed up. During back-ups, the bounds, f_l and f_u, and the lower and upper bound policies π_l and π_u are updated. Instead of performing value iteration until convergence as in LAO*, only a single back-up is performed over the set of all ancestors of the newly expanded states, $ancestors(S_{expand})$. If $fringe(G'_{\pi_l})$ is empty, the states in G'_{π_l} are backed up until either the estimated error of the initial state $\hat{\epsilon}(s_0) \leq \bar{\epsilon}$ or G'_{π_l} changes so that unsolved nodes appear among the leaves. States are never backed up twice in the same iteration. To speed up convergence, states far from the initial state are backed up first. The policy that is returned is the upper bound policy π_u where $V_{\pi_u}(s_0) \leq (1 + \hat{\epsilon}(s_0))V_{\pi^*}$.

1. $G' \leftarrow \{s_0\}$
2. **while** $\neg$timeout **do**
   (a) $\bar{\epsilon} \leftarrow \alpha \cdot \hat{\epsilon}(s_0)$
   (b) **while** $\hat{\epsilon}(s_0) > \bar{\epsilon} \wedge \neg$timeout **do**
      i. **if** $fringe(G'_\pi) \neq \emptyset$ **then**
         $S_{expand} \leftarrow$ subset of $fringe(G'_\pi)$
         for each $s \in S_{expand}$ **do** $expand(s)$
         $S_{backup} \leftarrow ancestors(S_{expand})$
      else
         $S_{backup} \leftarrow G'_\pi$
      ii. **for each** $s \in S_{backup}$ **do** $backup(s)$
3. **return** π_u

Figure 2: Iterative Bounding LAO*.

4.3 Evaluation functions

IBLAO* maintains lower and upper bounds of the optimal expected cost for each state s in the explicit graph G'. The current values of these bounds are denoted by $f_l(s)$ and $f_u(s)$, respectively. The lower and upper bound policies π_l and π_u corresponding to these evaluation functions are defined as follows:

$$\pi_l(s) = \arg\min_{a \in A} T_a f_l(s), \qquad \pi_u(s) = \arg\min_{a \in A} T_a f_u(s).$$

Every time a new unvisited state is added to G', its bounds are initialized using two heuristic functions: $f_l(s) = h_l(s)$ and $f_u(s) = h_u(s)$. These heuristics are assumed given as part of the problem and must satisfy $h_l(s) \leq V_{\pi^*}(s)$ and $h_u(s) \geq V_{\pi^*}(s)$ for all states s.

When a state is backed up, new bounds $f'_l(s)$ and $f'_u(s)$ are calculated from the previous f-values as follows:

$$f'_l(s) = \max\left(f_l(s), T_{\pi_l(s)} f_l(s)\right) \quad (4)$$

$$f'_u(s) = \min\left(f_u(s), T_{\pi_u(s)} f_u(s)\right) \quad (5)$$

The bounds guarantee that there *exists* a policy π such that $f_l(s) \leq V_\pi(s) \leq f_u(s)$. However, this does not tell us how such a policy can be found.

Theorem 1. If the upper bound heuristic h_u is *uniformly improvable*, i.e. for all states s

$$h_u(s) \geq \min_{a \in A} T_a h_u(s), \quad (6)$$

then the value function of the upper bound policy V_{π_u} is bounded by f_l and f_u, so that for all states s $f_l(s) \leq V_{\pi_u}(s) \leq f_u(s)$.

Proof. Since $f_l(s) \leq V_{\pi^*}(s)$, we also have that $f_l(s) \leq V_{\pi_u}(s)$. Assume that

$$f_u(s) \geq \min_{a \in A} T_a f_u(s). \quad (7)$$

Then after applying (5) on a state s', $f'_u(s') = T_{\pi_u(s')} f_u(s') \geq \min_a T_a f'_u(s')$ and for all other states s, $f'_u(s) \geq \min_a T_a f_u(s) \geq \min_a T_a f'_u(s)$. Since f_u is initialized with h_u, the condition (6) implies that (7) holds. Let $f_0, f_1, \ldots$ be functions such that

$$f_i(s) = \begin{cases} V_{\pi^*}(s) & \text{if } i = 0 \text{ or } s \text{ is a goal state,} \\ T_{\pi_u(s)} f_{i-1}(s) & \text{otherwise.} \end{cases}$$

This corresponds to the value function of a policy where actions are chosen according to π_u until i steps into the future when actions are chosen according to π^*. As $i \to \infty$, $f_i(s) \to V_{\pi_u}(s)$. If $i > 0$ and $f_{i-1}(s) \leq f_u(s)$, then using (7) $f_i(s) \leq T_{\pi_u(s)} f_u(s) \leq f_u(s)$. Because $f_0(s) = V_{\pi^*}(s) \leq f_u(s)$, it follows that $f_i(s) \leq f_u(s)$ for all i. □

Theorem 1 guarantees that the cost of the upper bound policy is always less than or equal to $f_u(s)$ for all s. No such guarantee exists for the lower bound policy. Also, since we have bounds on V_{π_u}, the final value iteration step of LAO* is not needed.

4.4 Error bound

A state is considered solved if the error of the expected cost of the upper bound policy relative to the expected cost of the optimal policy is smaller than the error bound $\bar{\epsilon}$:

$$\epsilon(s) = \frac{|V_{\pi_u}(s) - V_{\pi^*}(s)|}{V_{\pi^*}(s)}. \quad (8)$$

The optimal expected cost is not known, but using Theorem 1, we can bound the relative error with an estimate $\hat{\epsilon}$:

$$\hat{\epsilon}(s) = \frac{f_u(s) - f_l(s)}{f_l(s)} \geq \frac{V_{\pi_u}(s) - V_{\pi^*}(s)}{V_{\pi^*}(s)} = \epsilon(s).$$

When all successor states of a state s are considered solved, s will also be considered solved after being backed up.

Theorem 2. Let s be a state and let $\hat{\epsilon}(s') \leq \bar{\epsilon}$ for all $s' \in succ(s, \pi_l(s))$. Then backing up $s$will ensure that $\hat{\epsilon}(s) < \bar{\epsilon}$.

Proof. By (4) and (5), we have that $f'_l(s) \geq T_{\pi_l(s)} f_l(s)$ and $f'_u(s) \leq T_{\pi_u(s)} f_u(s) \leq T_{\pi_l(s)} f_u(s)$ for all states s. Since $\hat{\epsilon}(s') \leq \bar{\epsilon}$ for all $s' \in succ(s, \pi_l(s))$, $f_u(s') \leq (1 + \bar{\epsilon}) f_l(s')$ and thereby $f'_u(s) \leq (1 + \bar{\epsilon}) T_{\pi_l(s)} f_l(s) - \bar{\epsilon} q(\pi_l(s), s)$. Finally,

$$\hat{\epsilon}(s) = \frac{f'_u(s) - f'_l(s)}{f'_l(s)} \leq \bar{\epsilon} \frac{T_{\pi_l(s)} f_l(s) - q(\pi_l(s), s)}{T_{\pi_l(s)} f_l(s)} < \bar{\epsilon}. \quad \square$$

When $fringe(G'_{\pi_l}) = \emptyset$, the estimated error in all leaves of G'_{π_l} is less than or equal to $\bar{\epsilon}$. In this case, if the error bound has not converged so that $\hat{\epsilon}(s_0) \leq \bar{\epsilon}$, repeated back-ups of all the states in G'_{π_l} will either cause $fringe(G'_{\pi_l}) \neq \emptyset$ or, by Theorem 2, cause $\hat{\epsilon}(s_0) \leq \bar{\epsilon}$.

When $\hat{\epsilon}(s_0) \leq \bar{\epsilon}$ the inner loop is exited and the error bound $\bar{\epsilon}$ is reduced by a factor α where $0 < \alpha < 1$. The algorithm restarts at step 2b and expands states previously considered solved on the fringe of G'_{π_l}.

4.5 Expanding the fringe

Since Iterative Bounding LAO* does not use trials like many RTDP-based algorithms, the fringe may become very large. In each inner iteration, the algorithm therefore only selects a subset S_{expand} of the states in $fringe(G'_{\pi_l})$ for expansion.

Ideally, the algorithm should select those states whose expansions would have the largest impact on the estimated error of the initial state. Omitting such states may lead to unnecessarily many back-ups, while including other states leads to unnecessary work during expansion. A possible measure of this impact is the product of the estimated error in a state and the likelihood that the state will be reached from s_0 in the solution graph G'_{π_l}.

Since calculating exact state likelihoods is computationally expensive, we use an approximation $\hat{p}(s)$. The calculation of this approximation is interleaved with the calculation of the fringe itself as shown in Figure 3, and does not increase the computational complexity of finding the fringe. We then select those states that have an impact over average:

$$\hat{\epsilon}(s)\hat{p}(s) \geq \sum_{s' \in G'_{\pi_l}} \hat{\epsilon}(s')\hat{p}(s') \Big/ |G'_{\pi_l}| \tag{9}$$

Initially $\hat{p}(s) = 0$ for all states $s \neq s_0$ and $\hat{p}(s_0) = 1$
$queue \leftarrow \{s_0\}$
$fringe \leftarrow \emptyset$
while $queue \neq \emptyset$ **do**
  $s \leftarrow removefirst(queue)$
  for $s' \in succ(s, \pi_l(s))$ **do**
    $\hat{p}(s') \leftarrow \hat{p}(s') + \hat{p}(s)P(s'|s, \pi_l(s))$
    if $\hat{\epsilon}(s') > \bar{\epsilon}$ **then**
      if s' has successors **then**
        if $s' \notin queue$ **then** add s' to $queue$
      else
        add s' to $fringe$
  end for
end while

Figure 3: Algorithm for calculating the set $fringe(G'_{\pi_l})$ and the likelihoods $\hat{p}(s)$ for all states $s \in fringe(G'_{\pi_l})$.

4.6 Weighted heuristics

Just as with A* and LAO*, weighting the heuristic allows Iterative Bounding LAO* to make a trade-off between solution quality and the size of the explored search space. A separate evaluation function f_w is used for the weighted heuristic. For unexpanded states s, $f_w(s) = wh_l(s)$, where the weight $w > 1$. Using this evaluation function, a third policy π_w is defined where

$$\pi_w(s) = \arg\min_{a \in A} T_a f_w(s)$$

When a state s is backed up, f_w is updated as $f'_w(s) = T_{\pi_w(s)} f_w(s)$.

During search, instead of expanding states in G'_{π_l}, states are expanded from the solution graph of the weighted policy G'_{π_w}. When the weight is high, policies with many fringe states close to the goal where the heuristic estimates are smaller will be chosen before less explored policies. This reduces the size of the search space, but may cause optimal solutions to be missed. As with LAO*, in the worst case, the algorithm may converge towards a solution that is suboptimal by a factor w, and for all states s,

$$f_w(s) \leq wV_{\pi^*}(s). \tag{10}$$

The error bounds in states are estimated with the weighted estimated error $\hat{\epsilon}_w$, where

$$\hat{\epsilon}_w(s) = \frac{f_u(s) - f_w(s)}{f_w(s)}.$$

Theorem 3. If

$$\hat{\epsilon}_w(s) \leq \frac{\bar{\epsilon} + 1}{w} - 1 \tag{11}$$

holds, then the relative error $\epsilon(s) \leq \bar{\epsilon}$.

Proof. Using Theorem 1 and (10),

$$\hat{\epsilon}_w(s) = \frac{f_u(s) - f_w(s)}{f_w(s)} \geq \frac{V_{\pi_u}}{wV_{\pi^*}} - 1.$$

Then using (11) and (8),

$$\frac{\epsilon(s) + 1}{w} - 1 \leq \frac{\bar{\epsilon} + 1}{w} - 1. \qquad \square$$

Theorem 3 makes it possible to choose a weight $w \leq \bar{\epsilon} + 1$ such that when a solution is found in G_{π_w} the relative error is still less than or equal to $\bar{\epsilon}$. There is some freedom in how the weight w may be assigned. If $w = \bar{\epsilon}+1$, a state s will not be considered solved until $\hat{\epsilon}_w(s) = 0$, forcing the algorithm to expand every state in G_{π_w}. We use $w = \sqrt{\bar{\epsilon} + 1}$, which ensures that search branches can be pruned when $\hat{\epsilon}_w(s) \leq \sqrt{\bar{\epsilon} + 1}$.

When the error bound $\bar{\epsilon}$ is decreased after the inner loop of Iterative Bounding LAO* has completed, the value of the weight is updated as $w = \sqrt{\bar{\epsilon} + 1}$. In the next iteration, the explicit search graph G' cannot be reused directly because (10) only holds for the previous value of w.

In each state s we store the value $w(s)$, which is the value of w used by the algorithm the previous time s was visited. Let $G^w = \{s \in G' : w(s) = w\}$ where w is the current weight. Any other state $s' \notin G^w$ will be considered unexpanded. However, the information in G' is kept. Therefore, if a state $s \in fringe(G'_{\pi_w})$ that is to be expanded in G^w already is expanded in G', the old values of $f_l(s)$ and $f_u(s)$ are reused and the new value of the weighted evaluation function $f'_w(s)$ is computed as follows:

$$f'_w(s) = \max\Big(\frac{w}{w(s)} f_w(s), f_l(s)\Big).$$

5 EVALUATION

The new algorithm is evaluated against three other algorithms that use two-sided bounds: FRTDP, BRTDP, and VPI-RTDP, described in Section 3. We have also compared with ILAO* which is a more efficient version of LAO*. We have altered ILAO* so that it also updates the upper bound during back-ups. The main contributors to the total computation time are the number of back-ups and state expansions. The cost of performing a back-up is dependent on the branching factor of the problem while the cost of expanding states also depends

on the difficulty of computing the next state. We evaluated the algorithms using two benchmark problems which have been chosen for their difference in expansion costs.

The first benchmark problem is the racetrack domain which is a common benchmark problem for stochastic shortest path problems. This domain has been used for empirical evaluations of many algorithms similar to Iterative Bounding LAO* [6, 8, 12, 11]. Characteristic for this problem is that the branching factor is low and that new states can be generated quickly.

The second problem is troubleshooting of a printer which we believe is a suitable domain for probabilistic planning algorithms using two-sided bounds. A fault in a printer with an unknown fault should be repaired at a minimal expected cost. Probabilistic dependencies between observations and component faults are modeled with a Bayesian network. This is a realistic example problem where meaningful lower and upper bounds can be computed. Characteristic for this problem is a high branching factor and that expansions are expensive since they involve inference in a Bayesian network.

5.1 Racetrack

The racetrack domain was first presented in [1]. The task is to drive a vehicle from a starting position across a goal line. The states are integer vectors $s = (x, y, \dot{x}, \dot{y})$ describing the vehicle's position and velocity in two dimensions. Actions are integer accelerations $a = (\ddot{x}, \ddot{y})$ where $|\ddot{x}| \leq 1$ and $|\ddot{y}| \leq 1$. The states are fully observable, and uncertainty is introduced when actions are performed. If a wall is hit, the vehicle is moved back to the starting position. New states are easily computed involving only a simple check if the path between two points is blocked.

We have used two racetrack maps, *large-b* and *block80*, that have been published in [1] and [11] respectively. The problems are specified identically as in [12] where in *large-b* actions may fail causing the vehicle to skid and have zero acceleration and in *block80* a perturbing gust of wind may accelerate the vehicle in a random direction. The probability with which an action fails is 0.1. For non-goal states s, the lower bound heuristic used is

$$h_l = h_{\min}(s) = \min_{a \in A} \Big(q(a, s) + \min_{s' \in succ(a,s)} h_{\min}(s') \Big),$$

and for goal states it is zero. This is the optimal cost if action outcomes could be chosen freely. This heuristic has been used for this problem in [2, 12]. The upper bound heuristic is a constant, 1000, for all non-goal states. This is a gross overestimate of the optimal expected cost. This heuristic is in general not uniformly improvable. By introducing a special "plan more" action with a cost of 1000 that takes the vehicle directly to the goal, Theorem 1 will be applicable.

For each algorithm in the experiment, values of the upper and lower bounds in the initial state are available at any time. When these values have converged such that their relative difference is less than a threshold ϵ the algorithm is halted and the time in seconds and the total number of expansions and back-ups are registered. Also, the actual cost of the current upper bound policy V_{π_u} is evaluated. A time limit is set to 300 seconds.

Two version of Iterative Bounding LAO* are tested, weighted (wIBLAO*) and unweighted (IBLAO*). Both uses $\alpha = 0.5$ and for wIBLAO* $w(\bar{\epsilon}) = \sqrt{1 + \bar{\epsilon}}$. BRTDP uses $\tau = 50$, FRTDP uses $\epsilon = 0.001$, $D_0 = 10$, and $k_D = 1.1$, VPI-RTDP uses $\alpha = 0.001$ and $\beta = 0.95$. These are the same values used in the empirical evaluations in [8, 12, 11].

The results are shown in Table 1. The best results in each category are in bold. Iterative Bounding LAO* requires more back-ups than the other algorithms but fewer states are expanded. This is an expected result because Iterative Bounding LAO* backs up all ancestor states to the expanded states while the RTDP algorithms only back up the states on the trajectory of the last trial. ILAO* must expand all states in a solution and for *block80* almost the entire state space is reachable under the optimal policy. Therefore it is not able to complete within the time limit. Iterative Bounding LAO* would have this problem too if it expanded all states on the fringe instead of using (9). The weighted version of Iterative Bounding LAO* is more restrictive with expansions but requires more back-ups because of the necessary weight adjustments.

5.2 Troubleshooting

In the troubleshooting problem [7], a fault in a printer should be discovered and repaired. The actions have different costs and each action may either repair a component or make an observation. The health of the components is not known. A state is therefore a probability distribution over possible component faults, a so called belief state.

The printer system is modeled with a Bayesian network with two types of variables: components and observations. The network models probabilistic dependencies between these variables. A component variable describes the health of the component and an observation variable describes something that can be observed. The Bayesian network used for this problem is publicly available in repositories such as the Bayesian Network Repository [5] by the name *win95pts*.

When an action a that makes an observation is performed, evidence is added to a variable in the Bayesian network. For value the variable may have, a new belief state s' is computed given the previous belief state s. The likelihood of that evidence is $P(s'|a, s)$. When an action that repairs a component is performed, evidence is removed from all descendants of that component variable. A new belief state is computed by moving probability mass from situations where the component is faulty to situations where it is non-faulty. A motivation for this can be found in [10]. After each repair, there is a mandatory check whether the printer is working properly. State expansions require inference in the Bayesian network to be made. This makes the expansion step for this problem much more time consuming than for the racetrack domain. The inference algorithm used is Variable Elimination [4].

This problem is similar to a POMDP [3] but it is not as difficult. The reason for this is that each repair actions can only be performed once and each observation can only be performed a limited number of times because only repair actions may remove evidence. This property makes the reachable belief state space discrete and finite and the problem is therefore suitably solved with algorithms such as LAO* and RTDP.

A natural lower bound for this problem is the solution to a relaxation where all components are fully observable. Then

$$h_l(s) = \sum_c P(c|s) q_c^r$$

where $P(c|s)$ is the probability that the component c is faulty in the state s and q_c^r is the cost of repairing c.

In [7] an upper bound heuristic for this problem is presented. This heuristic is derived from a stationary policy that is guaranteed to repair the printer. Some components are observable, meaning that it is possible to observe the health of those components directly. The components are ordered from 1 to n: $c_1, c_2, \ldots, c_n$. The observable components are observed in this order and if a component is faulty it is repaired. The components that cannot be observed are repaired

Table 1: Comparison of algorithms on the problems *large-b*, *block80*, and *win95pts*.

large-b	ϵ	V_{π_u}	*expansions*	*backups*	*time*
wIBLAO*	1.0	34.70	**502**	**13118**	**0.08**
	0.1	23.95	**2606**	108064	0.53
	0.01	23.31	**3743**	203323	1.00
	0.001	23.26	**4353**	286681	1.39
IBLAO*	1.0	39.34	2294	32272	0.22
	0.1	24.86	3381	56766	0.45
	0.01	23.45	3995	86356	0.53
	0.001	23.27	4706	120142	0.78
ILAO*	1.0	28.00	6102	67127	0.25
	0.1	23.28	9133	342745	0.74
	0.01	23.25	9884	811285	1.49
	0.001	23.25	9909	902720	1.64
BRTDP	1.0	28.24	4170	19552	0.16
	0.1	23.48	5527	**33800**	**0.23**
	0.01	23.27	6416	**48270**	**0.28**
	0.001	23.25	6800	**58586**	**0.33**
FRTDP	1.0	28.35	4527	31842	0.20
	0.1	23.61	5354	53242	0.30
	0.01	23.27	6565	76546	0.38
	0.001	23.25	7246	96844	0.47
VPI-RTDP	1.0	27.67	4301	25750	0.19
	0.1	23.63	5357	57528	0.31
	0.01	23.29	6053	98088	0.44
	0.001	23.25	6768	160680	0.66
block80	ϵ	V_{π_u}	*expansions*	*backups*	*time*
wIBLAO*	1.0	14.10	**2217**	**17768**	**0.47**
	0.1	9.81	**10898**	157913	3.38
	0.01	9.61	18642	321675	6.51
	0.001	9.59	**24594**	481827	9.30
IBLAO*	1.0	15.57	5275	62998	1.55
	0.1	10.04	12576	227177	4.55
	0.01	9.61	**17232**	318582	6.31
	0.001	9.59	25614	475370	9.42
ILAO*	1.0	-	-	-	-
BRTDP	1.0	12.28	10574	46468	1.44
	0.1	9.66	21270	110288	2.95
	0.01	9.59	33423	193632	4.88
	0.001	9.59	41830	**270170**	6.55
FRTDP	1.0	11.91	9740	59916	1.38
	0.1	9.65	26985	175120	3.75
	0.01	9.59	41795	295436	5.88
	0.001	9.59	56126	447364	8.08
VPI-RTDP	1.0	12.51	9950	44584	1.38
	0.1	9.69	20490	**107640**	**2.91**
	0.01	9.59	32553	**192490**	**4.77**
	0.001	9.59	41058	272936	**6.36**
win95pts	ϵ	V_{π_u}	*expansions*	*backups*	*time*
wIBLAO*	1.0	15.33	60	294	5.22
	0.1	13.22	**673**	6243	**55.2**
	0.01	12.84	**891**	10517	**76.2**
	0.001	12.84	**945**	11895	**80.4**
IBLAO*	1.0	15.33	**55**	**234**	**4.30**
	0.1	13.18	806	**4877**	67.2
	0.01	12.84	1163	**7434**	96.6
	0.001	12.84	1235	**8013**	101
ILAO*	1.0	13.22	288	2402	11.7
	0.1	12.84	3899	45077	139
	0.01	12.84	5950	74327	206
	0.001	12.84	6182	78255	208
BRTDP	1.0	14.96	240	586	13.4
	0.1	12.84	1975	5992	108
	0.01	12.84	3081	10034	158
	0.001	12.84	3222	10552	164
FRTDP	1.0	16.02	244	692	14.7
	0.1	13.39	2421	10016	141
	0.01	12.84	2585	11026	150
	0.001	12.84	2722	11566	154
VPI-RTDP	1.0	13.80	171	416	10.5
	0.1	12.84	2009	6096	111
	0.01	12.84	3097	9998	158
	0.001	12.84	3286	10764	166

immediately when it is their turn in the sequence. The expected cost of this policy can be computed analytically without expanding any states. The upper bound is the expected cost

$$h_u(s) = \sum_{i=1}^{n} \Big(\big(1 - \sum_{j=1}^{i-1} P(c_j|s)\big) q_i^o + P(c_i|s)(q_i^r + q_p^o) \Big) \quad (12)$$

where q_i^o is the cost of observing component c_i, q_i^r is the cost of repairing c_i and q_p^o is the cost of determining if any more faults remain. Please refer to [7] for more details on how (12) is derived and how the components are ordered.

The algorithms are run with the same settings as in the previous experiment and the results are shown in Table 1. Expansion costs clearly dominate the computation time. Like before, Iterative Bounding LAO* expands considerably fewer states than the other algorithms, and is therefore faster for all tested error bounds.

6 CONCLUSION

Iterative Bounding LAO* is an algorithm for ϵ-optimal probabilistic planning for stochastic shortest path problems. The algorithm uses two-sided bounds on the optimal expected cost which are iteratively narrowed. The way in which the algorithm weights the lower bound heuristic reduces the size of the search space. Compared to the other algorithms in the empirical evaluations Iterative Bounding LAO* expands significantly fewer states. This is shown to be beneficial on problems where state expansions are expensive such as a troubleshooting and repair problem for printers.

ACKNOWLEDGEMENTS

This work is supported in part by Scania CV AB, the Vinnova program Vehicle Information and Communication Technology V-ICT, the Center for Industrial Information Technology CENIIT, the Swedish Research Council Linnaeus Center CADICS, and the Swedish Foundation for Strategic Research (SSF) Strategic Research Center MOVIII.

REFERENCES

[1] A.G. Barto, S.J. Bradtke, and S.P. Singh, 'Learning to act using real-time dynamic programming', *Art. Int.*, **72**(1-2), 81–138, (1995).
[2] B. Bonet and H. Geffner, 'Labeled RTDP: Improving the convergence of real-time dynamic programming', in *Proc. of ICAPS'03*, (2003).
[3] A. Cassandra, L. Kaelbling, and M. Littman, 'Planning and acting in partially observable stochastic domains', *Art. Int.*, 99–134, (1998).
[4] R. Dechter, 'Bucket elimination: A unifying framework for several probabilistic inference', in *Proc. of UAI'96*. Morgan Kaufmann, (1996).
[5] G. Elidan. Bayesian Network Repository. http://compbio.cs.huji.ac.il/Repository/, 2001.
[6] E.A. Hansen and S. Zilberstein, 'LAO* : A heuristic search algorithm that finds solutions with loops', *Art. Int.*, **129**(1-2), 35–62, (2001).
[7] D. Heckerman, J.S. Breese, and Koos Rommelse, 'Decision-theoretic troubleshooting', *Communications of the ACM*, **38**(3), 49–57, (1995).
[8] H.B. McMahan, M. Likhachev, and G.J. Gordon, 'Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees', in *Proc. of ICML'05*, (2005).
[9] N.J. Nilsson, *Principles of Artificial Intelligence*, Morgan Kaufmann, San Francisco, CA, 1980.
[10] A. Pernestål, *Probabilistic Fault Diagnosis with Automotive Applications*, Ph.D. dissertation, Linköping University, 2009.
[11] S. Sanner, R. Goetschalckx, K. Driessens, and G. Shani, 'Bayesian real-time dynamic programming', in *Proc. of IJCAI'09*, (2009).
[12] T.Smith and R.Simmons, 'Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic', in *Proc. of AAAI'06*, (2006).