A Probabilistic Search Heuristic for Path Planning in Uncertain Environments

Jamieson Schulte JSCHULTE@CS.CMU.EDU
Center for Automated Learning and Discovery, 5000 Forbes Ave, Pittsburgh, PA 15213 USA

Abstract

Path planning in partially observable domains often involves search in a very large state space for an optimal sequence of actions. When the search space is large, it becomes necessary to use a *heuristic* to guide the search into regions of the space that will yield the best results. Currently, however, most search methods consider each state in isolation when evaluating which to explore, ignoring context provided by other regions of the search space. This paper describes a heuristic that improves the contingency plans generated by a search algorithm, using an estimate of the probability that each plan under consideration will be executed. The execution probability estimate biases the search to concentrate on plans for which there is greater expectation of a large reward if executed. In cases in which a plan is provably worse than alternatives, it can be eliminated from the search altogether, thus saving memory and computation. Experiments show that significant improvements in computational efficiency and plan quality can be gained over methods that do not make use of this information, but that the improved plans are not helpful in our application domain.

1. Introduction

This paper addresses the problem of efficiently surveying the large search space in a path planning problem using probabilistic inference about the value of potential actions. The application area of the research is the problem of finding a *contingency plan* for a mobile agent to reach a goal position in an uncertain environment. The environment has a graph-like structure in which an agent must travel between vertices, along edges, to one or more goal vertices $g \in G$. A key complication is that it is unknown to the agent in advance whether a given edge can actually be traversed. This information can only be discovered by attempting the action and observing the result. The domain has been labeled the Canadian Traveler's Problem (CTP) (Bar-Noy & Schieber, 1991). Improvements to planning methods for the CTP domain would assist in many application areas in which an agent must travel through a partially observable environment. Examples include robot navigation in dynamic, crowded or cluttered environments, and vehicle path planning where roads may be impassable due to traffic or physical damage.

We wish to maximize a time-discounted reward acquired upon reaching a goal by finding the contingency plan with the shortest *expected* path length to the goal. A contingency plan is related to a classical plan in that it represents a sequence of actions to perform. However each action can have an uncertain outcome (success or failure), so for each case a new branch of the plan must be formed, leading to a tree-like structure rooted at the start state and branching at each point of uncertainty. During planning, the *search tree* represents every plan under consideration and can have a very large branching factor, making it prohibitively large. In the domain under consideration, we use a search tree because the value of an action is governed by the values of both success and failure of the action. If an action fails, then an alternative action must be used, and any alternative must be explored before its value can be known. The search tree is used to propagate and store the relevant information discovered during search.

In search problems, a heuristic is typically used to favor exploration in areas of the state space that will yield better plans. In some cases it is sufficient for a heuristic to consider only properties of the state under consideration when evaluating it. However, when creating a contingency plan, it is important to explore parts of the search space that may be very undesirable, but into which the agent may be forced by its environment. When using a conventional search heuristic in such a stochastic setting, planners tend to ignore parts of the search space that, while unrewarding, affect the overall quality of any plan due to the need to consider negative outcomes of actions. This paper describes a heuristic that uses information gathered by all parts of the search tree to guide exploration. In particular, the *probability of execution* of each plan or plan fragment is estimated using bounds on its expected reward. This estimate is used to prioritize the search so that plans that have greater likelihood of being executed are developed in greater detail.

Experimental results show that the new algorithm, called Weighted BA*, generates plans that are substantially more complete. That is, they account for a greater number of outcomes resulting from uncertainty in the world. Despite improved completeness, simulation results show that an agent will perform no worse in our test environment using a simpler plan. Instead of using a complex plan that covers many contingencies, an equally-performing agent uses a sparser plan generated by a competing planner and replans any time that new information about the environment becomes available.

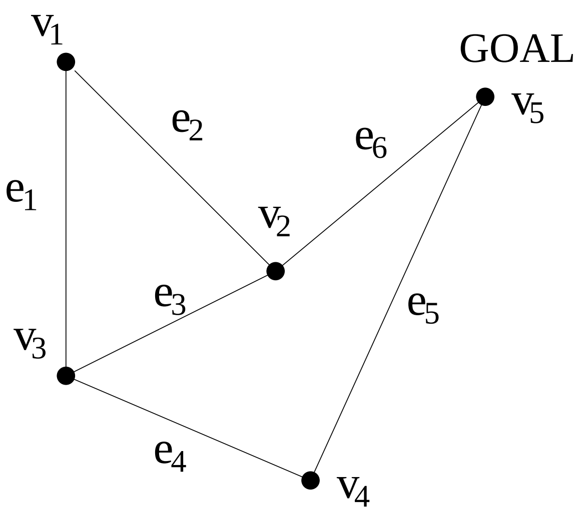

Figure 1. A simple graph environment. The agent's task is to travel from some starting vertex to a goal vertex along graph edges which are of unknown traversability.

2. Approach

A description of our approach to the problem requires some understanding of the Canadian Traveler's Problem and heuristic search algorithms. After each has been described below, we present the new planning approach.

2.1 Background

2.1.1 The Canadian Traveler's Problem

The Canadian Traveler's Problem asks how to most quickly travel from one city to another on a network of roads that may or may not be blocked by snow. To find out whether a road is passable, a traveler must attempt to drive on the road itself. More abstractly, we represent cities as vertices and roads as edges in a partially observable graph structure (as in Figure 1). The state of certain edges is set stochastically to either *true* or *false* (traversable or not traversable, respectively) according to some probability distribution, and never changes. An edge has a length, or cost, which is simply the Euclidean distance between vertices that it connects.

An agent in the environment can move between vertices along edges that are traversable. Moving on an non-traversable edge has the same cost, but brings the agent back to its original position. Initially, the state of one or more edges cannot be observed, and the agent will discover their values only if it attempts to move on them. In our formulation, the reward that an agent receives upon reaching the goal vertex is the time-discounted value:

$$r(t) = e^{-\alpha t}$$

where t is the time at which the goal is reached, and α is a reward decay factor that represents the degree of importance of reaching the goal quickly and is problem-specific. The time to reach the goal is proportional to the sum of the lengths of the edges used to get there. Since, to the agent, the world has hidden state and is not deterministic, a standard plan or simple sequence of actions is not sufficient to reach the goal. Instead a *contingency plan* is required, which branches at each point of uncertainty in the world. In this case, the reward to be received by following a contingency plan is unknown since it is uncertain how long it will take to arrive. Thus, we use a measure of plan performance based on expected reward:

$$E[r(t)] = E[e^{-\alpha t}] = \sum_{u \in T} P(u)R(u)$$

where T is the set of all possible goal-arrival times using the contingency plan and $P(u)$ is the probability of arriving at time u.

We assume that the agent has some prior belief $\hat{P}(e)$ about the state of each unobservable edge. If the agent is completely uncertain about an edge state, then a non-informative prior of $\hat{P}(e) = \frac{1}{2}$ is most appropriate. For simplicity, an agent assumes that edge states are selected independently in our approach, though this assumption is not critical to the method discussed here.

2.1.2 Heuristic Search, A*, and BA*

The search algorithm which we improve upon is an example of a heuristic search in belief space. (Lita et al., 2001) This algorithm, called Belief A* (BA*), is an extension of the popular A* planning algorithm(Nilson, 1982) to domains with partial observability. To explain the use of a priority function in heuristic search, we present a simplified version of the A* algorithm. (For brevity, the outline omits steps that record and check the shortest path to each state, which are used to eliminate suboptimal plans and to reconstruct the best plan in the final step.)

- Initialize an empty priority queue and add the start state to it.
- Repeat until the goal state is removed from the queue:
  - Remove the top-priority state s_{top} from the queue.
  - For each action available in s_{top} find the successor state s_i, assign a priority to it using the function $Pri(s_i)$ and put it into the queue.
- Find the path that led to the goal state and return this as the optimal plan.

The priority function in the A* algorithm has the form

$$Pri_{A*}(s) = d(s) + h(s)$$

where $d(s)$ is the distance along the path under consideration to state s, and $h(s)$ is the *heuristic function.* The heuristic function is an "optimistic" estimate of the distance from state s to the goal state. If $h(s)$ is always less than or equal to the actual distance to the goal, then it is called an "admissible heuristic," and A* can be proven to always find the shortest overall path to the goal while typically examining a small subset of the total space of possible plans. However, the better the estimate of the actual distance, the less searching that has to occur (if it were perfect, then there would be no need for search). A typical choice of $h(s)$ in path planning domains is the Euclidean distance from s to the goal state g. No path can be shorter than this, so it forms a valid lower bound, and it can be quickly computed.

The BA* algorithm extends this approach to uncertain domains. Rather than lower-bound the path length as in A*, $Pri_{BA*}(s)$ upper-bounds the expected amount that a branch of a plan passing through s will contribute to an agent's total reward. The change to priority based on expected reward is necessary because it is this value that is optimized in the CTP, not a deterministic path length. Here the priority function becomes:

$$Pri_{BA*}(s) = P(s)e^{-\alpha(d(s)+h(s))}$$

where $P(s)$ is the probability of reaching s following the plan up to that point, and $d(s) + h(s)$ can be recognized as the optimistic estimate of the distance to the goal state from the A* algorithm. By weighting the priority of states by $P(s)$, the search favors states that are more likely to be visited when the plan is executed. Under the assumption that the probability of success or failure of actions is independent, $P(s) = \prod_{e \in E_{prev}} \hat{P}(e)$, where E_{prev} is the set of edges previously visited to reach s, and $\hat{P}(e)$ is the agent's estimate of the probability that e is a traversable edge.

Since the domain of BA* is probabilistic, the algorithm does not terminate when a goal state is removed from the priority queue. The path that got us there is potentially just one of many contingencies that might occur. The BA* search continues until it is terminated after a set number of iterations, at which point the best contingency plan discovered so far is returned. In nontrivial worlds, there will likely be many contingencies for which the plan has not accounted, but the partial plan is expected to cover the most likely occurrences if it is provided with sufficient computation time. In order to generate a contingency plan after the search steps, a great deal of state information has to be stored by the algorithm. For each state visited, the algorithm records the parent and successor states, the agent's belief about the state of the world at that point, the length of the path to that state, and the probability of reaching it. Our current research focus is to find ways to minimize the amount of such information that must be stored during the search.

2.2 Weighted BA*

The contribution of our research is the Weighted BA* algorithm, which builds on the BA* approach for approximating an optimal solution to a CTP using limited computation and memory resources. First the *search tree* data structure built by Weighted BA* will be outlined, and then the method for using it to develop an improved heuristic will be shown.

2.2.1 Search Trees

In order to search the space of possible plans and then use the result of the search to determine which plan is best, intermediate information is stored which represents the candidate plans and their expected rewards. In Weighted BA*, the information in the search tree is additionally used to determine the relative quality of candidate plans, which influences the prioritization of search states associated with them. Figure 2 shows a small portion of a graph, an agent's position, and the subtree that would result from the two available actions. In practice, the tree grows to be very large, even in small environments, and this size is the main current limitation of the BA* algorithm.

The basic procedure for starting a search tree is to create a root node at the agent starting position. Each node represents a search state, which includes the agent position and also its belief about the state of the world. In Figure 2, we see that the agent starts at vertex $v1$ and has belief that edge $e1$ is traversable with probability 1, and $e2$ is traversable with 0.9 probability. When expanding the tree to include the action corresponding to $e1$, we see that a new child node is created in which the agent is at $v2$, but has the same belief about the world since nothing new was learned by taking that action. For the other action, things are more complicated. First, we see that there are two possible outcomes. In the first, the action succeeds, and the agent arrives at $v3$ and updates its belief to show that $e2$ is traversable. In the second case, the action fails and the agent returns to $v1$ knowing that $e2$ is down. The tree would next be developed by exploring actions available at the new leaf nodes, and the order in which new nodes are "expanded" is governed by the Weighted BA* priority function $Pri_{WBA*}(s)$.

2.2.2 Contingency Plan Weighting

The core idea behind plan weighting is to recognize that even when the actual expected reward of all actions from a state are not known, preference can be given to the exploration of actions that are more likely to be rewarding. The "preference" is encoded as a weight w_s which is given to each state in the search tree. In order to derive these weights, it is necessary to compare parts of the search tree, which is done by bounding the expected reward that they will generate.

Each state s in the search tree has an associated upper- and lower-bound on the expected reward that will follow from it, ub_s and lb_s. Important to the computation of these values are the quantities $P(s)$ which is the probability of reaching s given the plan it is a part of, and $d(s)$, the distance to s following the plan. A simple set of rules then shows how these bounds are set during the search:

Goal State If state s is at the goal vertex, then

$$lb_s = ub_s = P(s)e^{-\alpha d(s)}$$

where $d(s)$ is the distance from the start state to s. This is the precise contribution that s makes to the total expected reward.

Fail State If s is a failure state from which the goal cannot be reached, then

$$lb_s = ub_s = 0.$$

Nonterminal Leaf State If s a leaf state, but not at either goal or failure, then

$$lb_s = 0 \text{ and } ub_s = Pri_{BA*}(s).$$

Recall that $Pri_{BA*}(s)$ is an upper bound on the reward that an agent in state s can receive. The lower bound reflects that possibility that the goal cannot be reached from s.

Internal State If s is not a leaf state, then

$$lb_s = max_{c \in C} lb_c \text{ and } ub_s = max_{c \in C} ub_c$$

where C is the set of successor (child) states that may result from the actions available in s.

Each step above correctly preserves the upper and lower bounds of the affected states. When a particular action has uncertain outcome, it has two child states corresponding to success or failure of the action. In the last case above, we simplify this by treating the two outcomes as a single state c, in which lb_c is the sum of the lower bounds in the success and failure states, and ub_c is the sum of the upper bounds. These are mathematically valid expressions for the true lower and upper bounds on reward given the action.

Using a simple recursive function, it is possible to use the rules above to efficiently maintain the upper and lower bounds at all states in the search tree after the exploration of each new state. The reward-bounding information propagates only to parent nodes, and only if new information results in a change of bounds for the parent. It does not affect the bounds for child or sibling states.

Knowing the reward bounds now allows us to estimate the probability that any part of the plan tree will be used in the final contingency plan generated following the search. Consider Figure 3, which shows, for an

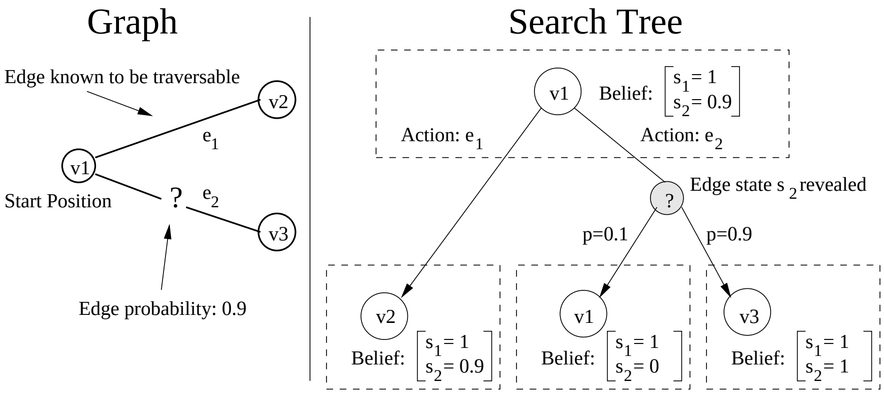

Figure 2. A pictorial representation of the search process. On the left is a very simple graph section with one edge of known state and another of unknown state. To the right is a diagram of the search tree for two possible actions available from vertex v_1. For the action that follows e_1, the agent arrives at v_2 with certainty and discovers no new information. The action that follows e_2, however, terminates with the agent on the edge having just learned whether it is traversable. In either case, the agent belief for s_2 is updated to reflect the new information.

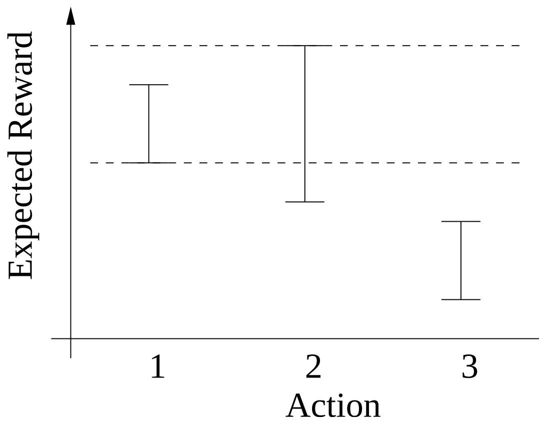

Figure 3. An example of bounds on expected reward for three actions available at one state. The dashed lines indicate the upper and lower expected reward bounds for the state itself.

example state, the bounds on expected reward for the available actions. Even though we do not know the exact expected reward for action 3, indicating that its corresponding subtree has not been fully explored, we can eliminate all further consideration of the action because its upper bound is below the lower bound on the state's expected reward. The comparison of states 1 and 2 is more subtle. To estimate the likelihood that one action will be chosen over the other, we need to consider the true expected reward for an action as a random variable, with a distribution that falls between the upper and lower bounds. For the example in Figure 3, let $f_1(r1)$ be the probability density function (pdf) for expected reward $r1$ for action 1, and note that $f_1(r1) = 0$ for all $r1$ outside of the bounded region. Also let $F_2(r2)$ be the cdf for the reward of action 2. Now it is possible to compute the probability that the actual reward of action 1 is greater than that of action 2:

$$Prob\{r1 > r2\} = \int_{-\infty}^{\infty} f_1(r) F_2(r) dr$$

Using the observation above and some form of probability distribution function over expected reward, it is possible to estimate for all states in a search tree the likelihood that one action will be more rewarding than another, even when their bounds overlap, as in action 1 and 2 in Figure 3. These estimates can be combined to evaluate the likelihood that any part of a plan in the search tree will be best, and thus end up being executed.

To do this, we give the root state of the search tree a weight $w_{root} = 1$, indicating that with probability 1, we know that one of the plans that pass through the root will be best. (This is obvious, since all of the plans start at the root state.) Then, for each child c of the root state, we set

$$w_c = w_{root} Prob\{r_c \text{ has the true highest reward}\}.$$

That is, we set each child's weight w_c to be a fraction of the parent's weight corresponding to the likelihood that it is the best action in that state. This procedure can then be applied recursively to each child until the entire tree as been updated. Note that the above operation touches every state in the search tree and is thus quite expensive.

Following an update of the weight values, we now have an estimate of the degree to which each contingency plan in the search tree is best. These weights are then used by the Weighted BA* priority function:

$$Pri_{WBA*}(s) = w_s P(s) e^{-\alpha(d(s)+h(s))}$$

In summary, the priority function gives priority to states that are part of plans that have higher weight, due to having greater likelihood of being the best plan. Unlike the BA* algorithm, which omits the w_s term from the priority function, the new approach aggregates information collected throughout the search tree in the form of expected reward bounds when prioritizing the search. Empirical results have shown that it is not helpful (and computationally expensive) to keep all weight values updated at all times. Best performance is observed when some number of search iterations occurs between updates, set by the parameter k.

The Weighted BA* algorithm proceeds as follows:

- Initialize an empty priority queue and add the start state to it.
- Repeat until a set number of iterations has occurred:
  - Remove the top-priority state s_{top} from the queue.
  - Calculate upper and lower bounds on the expected reward contribution of s and propagate these values to the parent nodes.
  - For each action available in s_{top} find the successor state(s) s_i, assign a priority to it using the function $Pri_{WBA*}(s_i)$ and put it into the queue.
  - If more than k iterations have elapsed since weights were last updated, update the weight values for the entire tree.
- Find the set of paths that result in the highest expected reward and return this as the best contingency plan that has been discovered.

Note that while the weighting of plans tends to focus computation on those that appear more promising, this is not a self-reinforcing process. Assume that a poor distribution $f(r)$ is used to model expected rewards, and an inferior plan receives large weights. The result of the increased computation spent developing the inferior plan will ultimately shrink its bounds, showing that it is comparatively poor. Upon re-weighting, the bulk of the weight will be distributed to other plans. Thus, the algorithm resists certain types of local maxima in the search space, though a poor reward model distribution $f(r)$ will slow it down.

3. Results

Two types of experiments have been performed to show that (1) the Weighted BA* algorithm generates more complete plans under computational constraints than BA*, and (2) the actual performance of agents using the new planner is better than that of previous methods. In the experiments that follow, the Weighted BA* algorithm used a uniform distribution between upper and lower bounds to model the expected reward. This distribution was chosen because it makes no assumptions about the reward value other than the valid assertion that it falls between the two bounds with probability 1.

In the experiments Weighted BA* performed a weight update every k iterations, where the initial value of k was 200, but increased by a factor of 1.5 with each update. As a result, updates occurred at iteration 200, 500, 950, 1625, and so on. For both planners, $\alpha = 0.005$, which has been shown to be appropriate for BA* in the environment.

3.1 Plan Completeness

While developing a *complete* contingency plan is not a necessity, it benefits an agent to be prepared for varying circumstances. A complete contingency plan is one that reaches a goal under all possible circumstances, declaring failure if and only if there is no path to the goal. In even mildly complex worlds, this is not a realizable option, but it is desirable to have a plan that covers the most probable scenarios. Thus, we can evaluate a plan based on how likely it is to provide a path to the goal considering all possible world states. [1]

We compared BA* and Weighted BA* planners in ten environments, selected randomly from a graph representation of the major roads in Honduras. Figure 4 shows the results of the plan completeness experiments. BA* and Weighted BA* were each tasked with developing a set of plans for the same set of 10 start states and 10 goal states. The plot shows the mean probability with which each planner could reach the goal, for a range of planning iterations. The error bars show values of one standard deviation above and below the mean. Typically, after approximately 10^5 iterations, the memory of a standard machine becomes nearly full.

In a typical BA* run, a single path to the goal is dis-

[1] In practice, an agent can re-plan if it reaches a state not accounted for in its current plan, so failure by a plan to reach the goal does not necessarily mean failure for the agent. This point is addressed using simulation experiments in the next subsection.

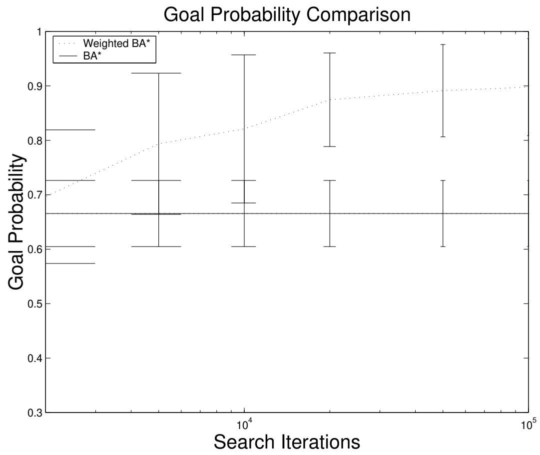

Figure 4. Comparison of the probability with which two planners can assure a path to the goal state.

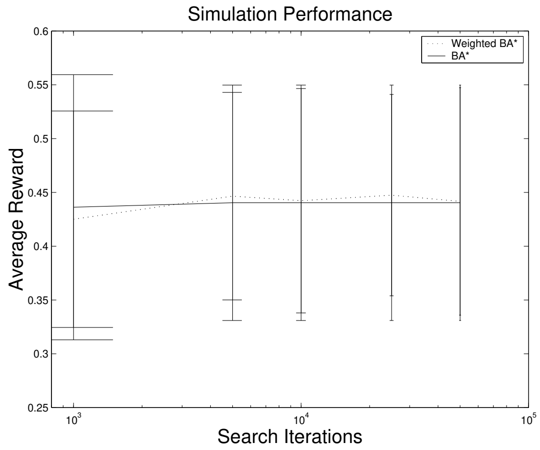

Figure 5. Comparison of actual agent performance in simulation.

covered after about 1000 iterations and further planning yields no improvements to that path. Weighted BA*, on the other hand, continues to find new paths as long as it iterates. After 10^5 iterations, between 5 and 20 paths to the goal state have typically been discovered, leading to a contingency plan that covers many possible states of the world.

3.2 Simulated Agent Performance

To test the actual performance of an agent using a planning algorithm, it is necessary to turn to simulation. The reason for simulation is that no contingency plan will cover all possible configurations of the environment. If an agent finds itself at an edge that it cannot traverse, and it has no alternative path in its contingency plan, then it replans from its new position and begins traveling again under the new plan. To test agent performance under these conditions we run them in a simulator and average results over many runs in randomized world configurations. The simulator uses road information from the country of Honduras consisting of 300 road segments (edges) and 234 intersections (vertices). Each bridge on an edge represents a point of uncertainty with 0.1 probability of not allowing the agent to pass. At the start of each simulation run, a world state is selected at random, with independence between bridges. In all cases, the agents have the correct prior $\hat{P}(e) = 0.9$, indicating that they know the true probability that a bridge will be traversable.

The experimental results show that the use of more complete plans has an insignificant influence on agent performance if replanning is allowed. In the simulation, an agent will replan if it encounters a state which is not part of its plan. For example, if it attempts to cross a bridge, fails, and has no contingency in the plan for what to do next, then it generates a new plan from its current position. Figure 5 shows a comparison between the BA* and Weighted BA* planners from 30 agent runs in randomized simulation experiments. The error bars show values of one standard deviation above and below the mean. Each planner experienced the same random set of tasks and environment configurations, thus while the variance is quite large, it is still possible to draw the conclusion that there is little performance difference between the two planners in this context.

4. Related Work

Little published research has been applied to planning in the Canadian Traveler's Problem domain. The BA* algorithm (Lita et al., 2001) described earlier is the only known algorithm specifically designed for planning when an agent has a prior estimate of edge state probabilities. The algorithm considers only a state's upper-bounded potential contribution to expected reward in its search heuristic, and thus explores many irrelevant states while ignoring states that will improve the plan that is eventually generated. The goal of the current research has been to reduce the amount of memory required by BA* to store the search tree, and to focus the search in areas that are of relevance to the planner. BA* is in turn based on the A* algorithm, a canonical approach to search in completely observable environments. Since A* cannot address contingencies resulting from failed actions, it is inappropriate in partially-observable situations if it is important to arrive at the goal state quickly.

The CTP can be cast as a very special case of a Partially Observable Markov Decision Process (POMDP), an area that has received a great deal of attention. In a POMDP, an agent is not fully aware of its current state, but can make observations from which it may be able to infer the state(Kaelbling et al., 1995; Littman, 1995). In general, the agent can only maintain a probability distribution over candidate states. To see this in the CTP domain, we represent the agent's knowledge of the state of the world as part of its current state (or belief state), because this affects its choice of action. (In any MDP, the state transition probabilities for an action are dependent only on the current state, which is the case here.) In the CTP, upon observing the effect of an action, the outcome of that action becomes certain to succeed or fail in all future cases. While general POMDP solvers exist and are the subject of widespread ongoing research, they tend to work only for a very small number of states (roughly 100)(Parr & Russell, 1995). In our problem, there are far more states, and there exist many additional constraints that make other approaches attractive. In particular, uncertainty in actions is always binary; the unknown state values (unobserved edges) are binary-valued and do not change after being observed; and aside from the unobserved edges, the agent is fully aware of its state.

A third research area that shares attributes with the Weighted BA* approach is search in stochastic game-playing. Here the goal is to select the best game move given uncertainty about the behavior of an opponent or the environment (in the form of rolled dice, for example) by building a search tree. (Baum & Smith, 1997) introduced the BP algorithm for game-playing domains which favors actions that have the largest mean in expected win probability. The method is best suited for game-playing environments, however, since the focus is on expected win probabilities, not on selecting actions with an associated cost to maximize a time-varying reward function.

5. Summary

A new algorithm, Weighted BA*, has been presented as a method of approximating an optimal contingency plan in the CTP domain using limited computational resources. The key idea behind the algorithm is that by aggregating information about the overall quality of subplans within the search tree, an improved heuristic can be derived. The new heuristic favors the exploration of states that develop plans which promise better reward, and thus are more likely to be executed. To determine which plans are best, upper- and lower-bounds on plan performance are propagated throughout the search tree from the leaf nodes. Periodically, every state in the tree is assigned a weight corresponding to the relative quality of the actions that lead to it.

Experimental results show that contingency plans generated with Weighted BA* cover far more of the possible states that an agent is likely to encounter. Tests in simulation, however, show that the improved plans do not necessarily lead to better performance by the agents when they are executed.

Acknowledgements

Sebastian Thrun and Lucian-Vlad Lita have been very helpful with this research, as have the KDD Lab reviewers. The DARPA CoABS program has provided funding for the work.

References

Bar-Noy, A., & Schieber, B. (1991). The canadian traveler's problem. *Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms* (pp. 261–270).

Baum, E. B., & Smith, W. D. (1997). A Bayesian approach to relevance in game playing. *Artificial Intelligence*, *97*, 195–242.

Kaelbling, L., Littman, M., & Cassandra, A. (1995). *Planning and acting in partially observable stochastic domains* (Technical Report CS-96-08). Brown University, Providence, RI.

Lita, L., Schulte, J., & Thrun, S. (2001). A multi-agent system for agent coordination in uncertain environments. *Proceedings of the 5th International Conference on Autonomous Agen ts.*

Littman, M. (1995). Solving partially observable markov decision processes via vfa. *Proceedings of the Workshop on Value Function Approximation, Machine Learning Conference 1995.*

Nilson, N. (1982). *Principles of artificial intelligence.* Tioga Publishing Company.

Parr, R., & Russell, S. (1995). Approximating optimal policies for partially observable stochastic domains. *Proceedings of the International Joint Conference on Artificial Intelligence.*