KDD Research Project: Planning for Single and Multiple Actors in Markov Decision Processes with Deterministic Hidden State

Jamieson Schulte
Center for Automated Learning and Discovery
Carnegie Mellon University
Pittsburgh, PA 15213
jschulte@cs.cmu.edu

January 16, 2002

Abstract

We propose a heuristic search algorithm for finding optimal policies in a new class of sequential decision making problems with single and multiple actors. This class extends Markov decision processes by a limited type of hidden state, paying tribute to the fact that many robotic problems indeed possess hidden state. The proposed heuristic search algorithm exploits the problem formulation to devise a fast bound-searching algorithm, which in turn cuts down the complexity of finding optimal solutions to the decision making problem by orders of magnitude. Furthermore, a method is developed for automatically computing a search heuristic based on the problem definition, rather than crafting one by hand. Extensive comparisons with state-of-the-art MDP and POMDP algorithms illustrate the effectiveness of our approach.

1 Introduction

The topic of decision making under uncertainty has received considerable attention in the past few years. Based on the recognition that uncertainty arises in a huge number of real-world problems, recent research has led to a range of paradigms for decision making under uncertainty. This project addresses the problem of planning for multiple actors in environments affected deterministically by hidden state, where there are a limited number of "consumable" goal states. A theoretical framework is developed that approximates the world faced by many physical "agents", which we model with Markov Decision Processes with Deterministic Hidden State (MDPDHS). Here the outcome of actions is predictable, given full knowledge of the state of the environment. In general, however, the actual state of the world can only be represented as a probability distribution over a finite set of possible configurations. We show that in some situations, optimal high-level planning can still occur, and that in many others approximations to the optimal solution can still be computed efficiently.

As an example of deterministic hidden state, a robot traveling to a destination in a dynamic office or battlefield environment may have a sufficient map, and a model of its kinematic behavior that renders its primitive actions predictable. Nonetheless, high-level uncertainty about the traversability of different paths may exist, due to closed doors or destroyed roads. The robot may then have to visit a location to discover its state. Such state is said to be "hidden" until it can be inferred by observation or other action, and is assumed in our framework not to change over time.

Our goal is to develop optimal or approximations to optimal *contingency plans* by explicitly representing this uncertainty during planning using the notion of *information state*. The information state is a probability distribution over the hidden state, and is used to determine the *expected* outcome of

actions. Importantly, as actions are performed and more of the environment is visited, the entropy of the distribution decreases monotonically. Note that such an approach is limited to situations in which some "prior" distribution over hidden state is available.

In the work described in this paper, individual actors use a new planning method to efficiently compute plans in environments described by a MDPDHS. Further experiments are presented in which multiple actors plan with a small amount of communication that allows limited coordination of actions. Coordination is necessary in the problem domain because we wish to maximize a global reward where each goal state can be reached by only one actor, raising the problem of contention between them. While the "multi-agent" aspect of the work has not been fully developed, it is shown that probabilistic information made available in information state planning can be useful when multiple actors must pursue a set of related goals in an uncertain environment.

In general, the acting systems that this work addresses are physical robots or humans acting under guidance in a large-scale logistical problem. What ties this work to the physical domain is that it is assumed a large amount of computation time and memory is available to compute plans. For tasks typical of software agents, decisions must be made quickly and with little computational overhead. Though there are likely to be exceptions to that generalization, our experiments have focused on physical scenarios.

This project report begins with a discussion of Markov Decision Processes and the generalized model that includes deterministic hidden state. Next, a method for planning with hidden state by using bounded estimates of reward given the observable state is described. In the multiple-actor case, we decompose the planning process into two interacting steps: individual planning, and coordination of multiple actors. The coordination step is simply a generalization of the individual planning step, and is treated after the single-actor planning has been fully described. Next empirical comparisons to related algorithms are reported, followed by a survey of related research.

2 Markov Decision Processes with Deterministic Hidden State

To make explicit some of the assumptions made by our planning method we use a model of the interaction between an actor and its environment. Today's two most prominent such paradigms are known as Markov decision processes (MDPs) [21] and partially observable Markov decision processes (POMDPs) [6]. While MDPs can cope with uncertainty regarding the effects of an agent's actions, POMDPs also address uncertainty in perception and uncertainty arising from hidden state.

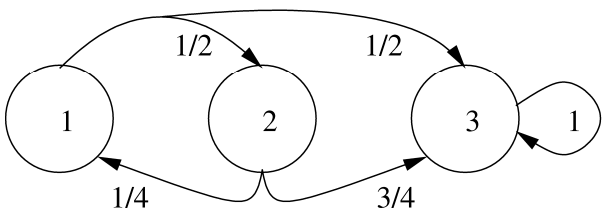
For many robotics domains, neither of these paradigms is well suited. MDPs are too specific in their assumption of full observability of the state of the environment. POMDPs, on the other hand, are so general that today's best algorithmic solutions can only cope with a dozen or so states. This raises the question as to whether there exist intermediate problem formulations, which address uncertainty yet allow for algorithms that scale better to larger state spaces.

The MDPDHS model is meant as such an example, which describes a very simple form of uncertainty that is nonetheless useful in practical situations. This section describes the MDPDHS framework, beginning with a brief review of MDPs.

2.1 "Classical" Markov Decision Processes

A Markov Decision Process (MDP) is a model that represents the interaction of an actor with its environment. It encodes the probabilistic outcomes of a finite set of actions (resulting from decisions on the part of the actor) occurring at discrete time steps. A MDP "state" is a collection of facts that may affect action outcomes. In robotics, this is typically the position of an actor, its internal state, and relevant information about the external world. The key assumption that makes the model computationally attractive is that the probabilistic outcome of an action depends only on the current state, independent of any previous states or actions. The resulting statistical independence of actions in states greatly simplifies the computation involved in decision making.

A MDP consists of the following components:



$$\begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 1: A diagram that shows the states of a simple three-state MDP and the possible transitions for only one action, u. In the matrix to the right, we see the transition probability matrix for this action where $p_{ij} = P(x' = j|x = i, u)$.

- **States.** The state of the MDP is denoted by x. At any point in time, the state is fully observable, meaning that there is no notion of partial observability, hidden state, or "sensor noise" in MDPs.
- **Actions and state transitions.** Actions, denoted by u, affect the state of the MDP in a probabilistic way. State transitions are characterized by a time-invariant conditional probability distribution $p(x'|u, x)$ that characterizes the probability of being in state x' after executing action u in state x.
- **Reward.** MDPs require a reward function $R(x) \longrightarrow \Re$, which measures the 'goodness' of each state x.

The central problem in MDPs is to devise a policy for action selection $\pi(x) \longrightarrow u$ that maximizes the expected cumulative future reward:

$$E[\sum_{t=0}^{\infty} \gamma^t R(x(t))] \tag{1}$$

Here $0 < \gamma \leq 1$ is a discount factor, which may be used to decay the reward over time. E denotes the mathematical expectation, and t is a time index. A well-known difficulty of finding such a policy arises from the fact that actions might affect rewards many time steps later. A number of methods are popular for calculating policies [11], including *value iteration* [21], *policy iteration* [23], or search in policy space using gradient descent [3, 13]. The relation to heuristic search was pointed out in [8]. However, the main deficiency of the MDP model in many real-world domains (such as robotics) is the requirement for fully observable state. Typically, a robot must make decisions without knowing the precise configuration of the environment. Will people be obstructing the next hallway? Will the door around the corner be open? These factors affect the outcome of actions deterministically, but cannot always be observed in all states.

2.2 Deterministic Hidden State

The MDPDHS model allows for the existence of a restricted type of hidden state. The nature of hidden state and its effect on state transitions and perception is limited in a way that allows for efficient, discrete problem solving, making the model similar to MDPs. In particular, it is not as general as the POMDP family of models [20, 6]. Nevertheless, as we shall see below, it is powerful enough to capture a range of practical problems with hidden state.

MDPDHSs extend MDPs as follows:

- **Hidden states**. In addition to the observable state x in MDPs, MDPDHSs possess hidden state, denoted by y. The state space of an MDPDHSs, thus, is the cross product of the observable state x and the hidden state y. By this it is meant that for each value of x, the full state could include any possible value of y. Initially, the hidden state is unknown to the actor. Instead, it is assumed to have a prior probability distribution p'_I over the values of the hidden state.
- **Hidden state transitions.** Hidden state transitions are deterministic—hence the term 'deterministic' in MDPDHSs. They are governed by a state transition function $\sigma(x, y, u) \longrightarrow y'$. In

addition, if the hidden state influences the transition of a fully observable state variable in x, this transition must also be deterministic (and such transitions must exist, otherwise the hidden state is irrelevant).

- **Measurements.** To infer information regarding the hidden state y, the actor can sense. Sensor measurements will be denoted by z. Measurements of state variables in x and y are also also deterministic, that is, they are governed by a function $\mu(x, y) \longrightarrow z$. The dependence of μ on x is important

It is easy to see that the MDPDHS model is a generalization of the MDP model. The key advantage of the MDPDHS model is that it allows for hidden state—which is paramount for many real world applications. However, from a technical point of view any MDPDHS can be transformed into an MDP of possibly exponential size (see below). By doing so, one loses important insights into the structure of the problem on which the individual planning algorithm described in this paper builds. We also note that the MDPDHS model is not as general as the POMDP model. For example, POMDPs allow for noise in perception and changes to the hidden state over time, whereas MDPDHSs do not.

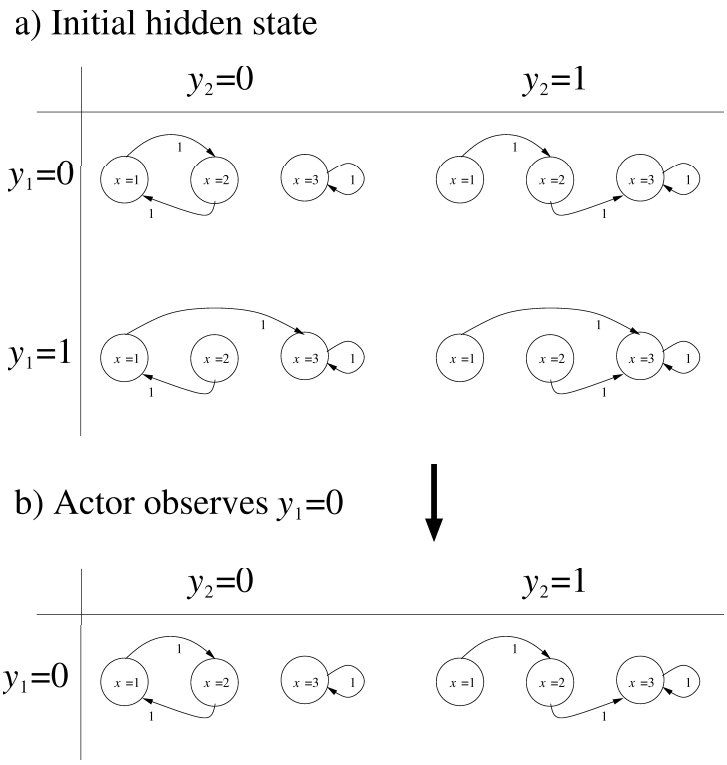


Figure 2: A diagram that shows the information state for a simple MDPDHS a) before, and b) after an observation is made. y_1 and y_2 represent the hidden state. The hidden state probabilities are updated following the observation that $y_1 = 0$, completely disallowing the possibility that $y_1 = 1$. As in this example, observations decrease the uncertainty in the problem, reducing the entropy of the hidden state distribution.

2.3 Information States in MDPDHSs

The key advantage of the MDPDHS model—which our algorithm below exploits—is the fact that the *information state space* is *finite*. The finiteness of the information state space implies that discrete search algorithms can be applied to find optimal policies in MDPDHSs.

Following [6], the *information state* is the state of knowledge an actor might have regarding the true state of the world, x and y. In the case of the observable state x, the corresponding part of the information state is simply equivalent to x. For the hidden state y, however, the corresponding part of the information state is a probability distribution over all possible states y [20]. This distribution will be denoted p_I, to indicate that it corresponds to an information state. The complete information state is thus the tuple $\langle x; p_I \rangle$.

As the agent acts and perceives, its information state changes. The observable state x changes stochastically according to the state space dynamics $p(x'|u, x)$. The change of the distribution p_I is captured by the popular Bayes filters [5], applied to the restricted state space dynamics in MDPDHSs. In particular, state transitions modify the information state as follows:

$$p_I(y = a) \quad \longleftarrow \quad \sum_b I[\sigma(x, b, u) = a]\, p_I(y = b) \tag{2}$$

Here x is the present observable state, u is the action, $I[\ ]$ denotes the indicator function, and the variables a and b instantiate possible hidden states y. Measurements z modify the information state as follows:

$$p_I(y = a) \quad \longleftarrow \quad \eta\, I[\mu(x, a) = z]\, p_I(y = a) \tag{3}$$

Here η is a normalizer that follows from Bayes rule, and ensures that p_I remains a proper distribution.

Figure 2 shows the transformation of the model of one action in a very simple MDPDHS that is similar in form to the MDP in Figure 1. (Because the transitions for only a single action are shown, it is not apparent from the diagram alone that this is a decision process.) The hidden state values y_1 and y_2 govern the transitions from states x_1 and x_2, respectively. Initially, the hidden state y is completely unknown, so there are 4 possible hidden state configurations, each a *deterministic* decision process. When the actor makes observation z_1, (perhaps by attempting the state transition governed by y_1) the space of possible hidden states is reduced, and the transition out of state 1 becomes a known, deterministic value.

At first glance, it might not be obvious why the information state space in MDPDHSs is finite. In principle, any distribution over $p_I(x)$ constitutes a valid information state with respect to the hidden variables y—and the space of all distributions if infinite. However, MDPDHSs owe the finiteness of their information state space to their deterministic dynamics and measurement functions relative to the hidden state y.

Why, then, is the information state space in MDPDHSs finite? Obviously, each information state p_I is defined through the values $\{p_I(y_1), p_I(y_2), \ldots, p_I(y_m)\}$, with m being the size of the hidden state space. Deterministic state transitions can only permute these values or add some of them together. For example, the two values $p_I(y)$ and $p_I(y')$ with $y \neq y'$ are added together if $\sigma(x, y, u) = \sigma(x, y', u)$. Consequently, deterministic state transitions can only lead to a finite number of successor distributions, since there are only finitely many permutations and there may only be a finite number of cases where two or more non-zero p_I-values are added together. A similar statement can be made for measurements. Each value $p_I(y)$ will either remain the same (before normalization with η) or be multiplied by 0. The latter is the case if $p_I(y) > 0$ but $\mu(x, y) \neq z$. The number of times in which a non-zero value $p_I(y)$ gets set to zero can only be finite. Thus, there are only finitely many distributions p_I that have to be considered. Notice that our argument exploits the fact that for deterministic hidden state, the number of non-zero p_I-value never increases.

We note, however, that the space of information states is generally much larger than the state space itself. If, for example, the number of hidden states y is m and each hidden variable can be in one of two states, then the number of information states that can be defined over y is $\Omega(2^m)$—although not all of them might plausibly occur. Nevertheless, any algorithm for computing a policy for action selection is likely to be dominated by the hidden state y, rather than the observable state x. In the next section, we will exploit this insight by using state space search to develop performance bounds for the much harder problem of finding solutions in information state space.

3 Single Actor Planning

Having introduced the MDPDHS framework, the question remains "how can we exploit the properties of MDPDHSs to generate contingency plans efficiently?" The answer, explained below, is to combine heuristic search with traditional MDP solutions.

3.1 Individual BA* Search

This section describes a search algorithm proposed for solving MDPDHSs which we will refer to as *BA** (short for *belief space A**). Our algorithm only applies to MDPDHSs that terminate, e.g., by reaching a goal state. At the core, our search algorithm is similar to the decades-old A* algorithm [14]. A primary difference is that BA* searches in the space of all information states, instead of in the space of all states. Just like many other modern-day implementations of heuristic search [18, 12], our approach can employ memoization with a hashing function to avoid searching the same information state twice—thereby reaping much the same benefit as, for example, value iteration does.

The key algorithmic innovation in this paper is a method for generating bounds that are utilized to prune the search tree and as a search heuristic when deciding what node to expand next. As we will demonstrate in our experiments, this mechanism can reduce the computation for finding a policy by several orders of magnitude.

3.1.1 Search in Information Space

At the core of the BA* search algorithm is a search tree, similar but not identical to trees found in the stochastic games literature [1]. The tree possesses two types of nodes: *choice nodes* and *response nodes* as shown in Figure 3 (see also [6]). Each of these nodes is labeled with an information state $\langle x; p_I \rangle$. Choice nodes represent situations where the actor can make an action choice. Since the agent maximizes its reward, it will pick the action u that maximizes the value of its children. Response nodes react to the action choice in a non-deterministic way. The non-determinism arises from two factors: The stochastic nature of the MDP, and the unknown hidden state y. Thus, the search algorithm averages the values of the response nodes, weighted in proportion to the probability of each succeeding information state.

Mathematically, the tree is used to recursively calculate an expected cumulative reward for each choice node in the tree. Let $V(x, p_I)$ denote the expected cumulative reward for being in the information state $\langle x; p_I \rangle$. $V(x, p_I)$ is defined recursively through the value of its children and their children:

$$V(x, p_I) \quad \longleftarrow \quad R(x) + \gamma \max_u \sum_y p_I(y) \sum_{x'} p(x'|x, y, u) V(x', p'_I) \tag{4}$$

where p'_I is the distribution over states y (i.e., the information state with regards to the hidden state y) that results by applying action u to the state $\langle x; y \rangle$ and the initial information state p_I. The equations for arriving at p'_I are given by (2) and (3). Thus, the search tree is a max-avg tree in information space, where choice nodes maximize the V-function, and response nodes average it.

The output of a MDPDHS search is a contingency plan that maps information states to the action that will provide the greatest expected future reward. Such a plan is tree-like in structure since at each response node, the outcome is uncertain - it is only the action taken at each choice node that is fixed in the final plan.

3.1.2 Propagating Bounds

The key innovation of the BA* search algorithm is a mechanism for calculating bounds on the V-value of each information state. These bounds are used to guide the heuristic search (which node to expand next) and they form also the basis for deciding which branches to prune entirely in the search. They are the key reason for the computational efficiency of our approach, which will be documented extensively in the experimental results section of this paper.

The intuitive idea is simple: Suppose we would like to know the V-value of an information state $\langle x; p_I \rangle$. Instead of performing the full search in information space, we can search the observable state

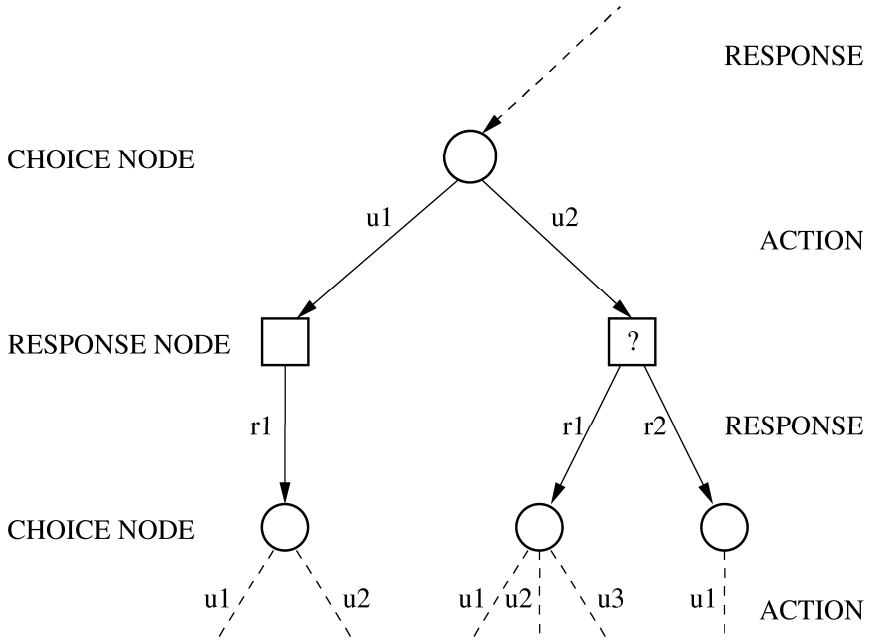


Figure 3: A diagram showing the structure of a portion of a search in an MDPDHS problem. At choice nodes (circles), the actor selects an action u, which will lead to a response node (squares). At a response node, the environment determines the outcome of the action, potentially subject to hidden state variables. The possibly unknown outcome leads to a new choice node, where the actor again selects an action.

`set-reward-bounds`(*node*)

- if *node* is a leaf:
  - $\alpha(node) \leftarrow$ `value-iteration-reward-lower`(*node*)
  - $\beta(node) \leftarrow$ `value-iteration-reward-upper`(*node*)
- if *node* is not a leaf:
  - $\alpha(node) \leftarrow max_{c \in C} \alpha(c)$
  - $\beta(node) \leftarrow max_{c \in C} \beta(c)$
    (where C is the set of children of *node*)

Table 1: Algorithm for calculating upper and lower bounds on the expected future reward for a node in the BA* search tree.

space x instead. This search is exponentially faster, since the state space is exponentially smaller than the information space (in most cases). To search in state space, we need to assume knowledge of the hidden state y. This raises the question as to what hidden state y shall we use in the search. The answer is straightforward: If at any point in time, we choose the hidden state that maximally beneficial to our goal of maximizing reward, the resulting value function will be an upper bound to the true V-value. Similarly, if we always assume the worst state y, the result will be a lower bound.

We will now formalize this idea. Let α be the desired lower bound, and β the desired upper bound. Then the bounds are defined recursively as follows:

$$\alpha(x, p_I) \quad \longleftarrow \quad R(x) + \gamma \max_u \min_{y: p_I(y) > 0} \sum_{x'} p(x'|x, y, u) \alpha(x', p_I)$$

$$\beta(x, p_I) \quad \longleftarrow \quad R(x) + \gamma \max_u \max_{y:p_I(y)>0} \sum_{x'} p(x'|x, y, u)\beta(x', p_I) \tag{5}$$

where p_I is the present information state regarding the hidden state y. It is straightforward to show these values bound the V-function:

$$\alpha(x, p_I) \quad \leq \quad V(x, p_I) \quad \leq \quad \beta(x, p_I) \tag{6}$$

Notice that for any fixed p_I, these bounds are calculated entirely in the state space of the observable state x, and not in the hidden state y or the more complex information state space. Thus, they can be found very efficiently. In our implementation, they are calculated using value iteration, and their calculation consumes a negligible fraction of the overall search time.

We also notice that these bounds are different from the familiar alpha beta bounds in two player game search. Those bounds are only applicable to min-max search. Our bounds exploit the duality between information space and state space, and they are calculated using a separate search algorithm. However, we conjecture that they are just as useful in searching MDPDHSs as the alpha beta bounds are in solving two-player games.

3.1.3 Search Heuristic

The performance bounds play a key role in determining which information state to expand next, and which states can be pruned from the tree altogether.

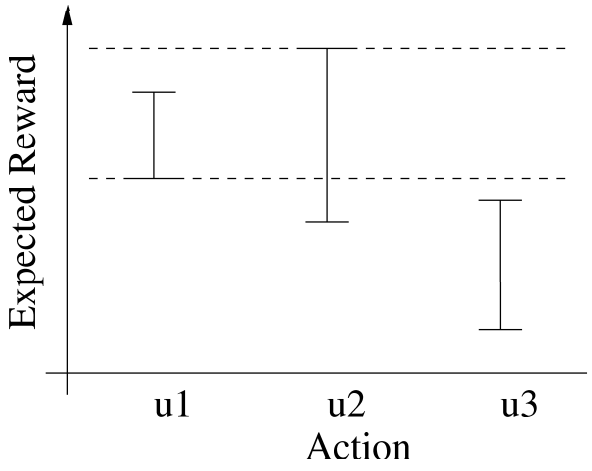


Figure 4: A comparison of bounds on reward for three actions available at some state $\langle x, p_I \rangle$. The dashed lines represent the upper and lower bounds on reward for the state itself, assuming that the action that maximizes the lower bound is selected (after possible additional planning that could further shrink the bounds.)

`prob-of-maximal-reward`(*node*,*action*)

- return the probability that taking $action$ in the state represented by $node$ would yield the maximum reward of the available actions, given their expected reward bounds and the assumed distribution over expected rewards.

Table 2: Algorithm for determining the probability that an action is best in a state, used to find action weights w.

The purpose of a heuristic function in heuristic search is to select the order in which nodes in the growing search tree are to be *expanded*. (Expansion is the process of examining a choice node and adding its children corresponding to each action possible at that point.) Available search states

are prioritized by the heuristic in such a way that the total number of nodes that require expansion is minimized, thus minimizing the computational requirements of the algorithm. In A* search, the heuristic values states according to a hand-coded function that estimates their value. In BA* search, we use the α and β bounds to estimate the value of states *automatically* from the problem definition.

A key difference between the A* and BA* algorithms is the latter operates in domains with hidden state, thus requiring that a contingency plan be generated, rather than a simple path through a known environment. As a result we wish to know which information states are *most likely* to be visited when the plan is executed, in addition to their expected value, when selecting the ordering in which they are expanded. However, without knowing what the final contingency plan will be, we cannot know with what likelihood each state will be visited. We next propose a method for estimating these probabilities with what information is available during the search process.

As a first step, we estimate the probability that a given branch from a choice node will in fact maximize the expected V-value from that state. To know the true expected reward for an action, it would be necessary to perform a complete search of its subtree, so our goal is to estimate the probability with which each branch will maximize the V-value using only the existing bounds available at the time of state expansion.

Consider the case of a state with n actions under consideration: $u_1, u_2, ..., u_n$. For each choice of action u_c, let $V^c_{x,I}$ be a random variable representing the unknown actual value $V(x, p_I)$ for the state under the condition that action c is taken. We wish to compute $G^c_{x,I}$, the likelihood that action u_c will yield the greatest reward:

$$G^c_{x,I} = Pr\{V^c_{x,I} \ \geq \ V^k_{x,I} \ \forall \ k \neq c\}$$

To compute this value exactly requires knowledge of the distributions governing each of the random variables $V^\bullet_{x,I}$, but these cannot be known with searching each action's subtree completely. Instead, we posit the use of a simple distribution, bounded by the known α and β values, to represent each reward distribution in the computation. Given an appropriate set of distributions, it is then straightforward to compute the likelihood with which each is most likely to yield the greatest reward value. Table 2 summarizes the step of computing the G-value of a state.

The choice of reward distribution is governed by the constraint that plausible reward values all have non-zero probability. Furthermore, it is desirable that reward values that cannot possibly occur have zero probability. A natural choice, then, is a function that is bounded by α and β. In the absence of additional evidence, a uniform$[\alpha, \beta]$ distribution is used in our work, though other distributions may make sense in other contexts in which there is reason for statistical bias.

Having computed the probability G^c_{x,p_I}, we can label each edge out of a choice node with an estimate of the probability that the corresponding action will be selected from that state when the plan is executed. The overall goal, however, is to develop a search heuristic that takes into account the likelihood of each state being visited under a reward-maximizing plan. Since these values are only rough estimates of the actual probabilities, we will hereafter simply refer to them as *weights* and use the notation w_{x,p_I} to represent the weight of node representing the information state $\langle x, p_I \rangle$. We can estimate weights by labeling all choice edges in the search tree with a G-value and assigning a weight of 1 to the root node, where all plans start (since it actually does have probability 1 of being visited). The children of the root node can then be said to have a weight equal to the G value of the corresponding edge.

In general, the child of a choice node is assigned a weight equal to the product of the parent's weight and the edge connecting them. Children of response nodes are assigned weights equal to the product of the parent's weight and the probability of the edge connecting them, which comes directly out of p_I, and is very simple to compute if the components of the hidden state y are statistically independent.

Weight values change as the tree expands and the reward bounds shrink, thus altering action reward distributions and the G-values that depend on them. Updating weight values is typically efficient, however, since a node's weight need only be updated if the α or β bounds of an ancestor or an immediate child change.

Finally, the weight values are used directly as the heuristic function that sets the order of node expansion. Unexpanded nodes with greater weight value are expanded before those of lesser value. We note again that our approach differs from A* in that the heuristic is not supplied by a human designer. Instead, it is derived automatically from the problem description.

3.1.4 Pruning

Consider a choice node. Here we choose the action u that *maximizes* the expected V-value of its sub-tree. Clearly, if the β-value of a sub-tree is smaller than the α-value of another, it is impossible that this subtree corresponds to the optimal action and hence it can be pruned. Stated in terms of the search heuristic, if the reward distribution of an action falls strictly below that of another action, then it will have zero probability of being executed and will be assigned a zero G-value. Thus all of its descendant nodes will have weights equal to zero, and can be pruned immediately. The benefit of pruning is that significant portions of the tree can be removed from memory if this fact is discovered sometime after the nodes have been expanded, and all future examination of that portion of state space can be eliminated.. As we will see below, pruning trees in this way can dramatically reduce the overall computation and memory requirements.

3.1.5 "Anytime" Planning

Unfortunately, even using the BA* search heuristic and tree pruning, it is impractical to perform a complete search in many environments due to time and memory limitations. It is still possible to generate a contingency plan that is not strictly complete in the sense that all configurations of hidden state are accounted for. In this sense, the BA* algorithm has the "anytime" property that it can be stopped at any time and queried for the best contingency plan given an incomplete search. In many cases, the resulting plan will cover the most likely hidden state configurations, and provide relatively tight bounds on the expected reward for the incomplete plan. The algorithm for recovering a partial plan from an incomplete search tree is described next.

Since the search tree at any moment during planning represents the bounded expected outcomes of visited states, it is straightforward to generate a plan that has the greatest expected reward. The action weighting values w, however, are no longer useful to determine the reward, since we now know that only the action with the best expected outcome will be selected, and the weights' purpose was to represent uncertainty in action choice. To generate a contingency plan then, the best action is selected at each node, starting at the leaves of the search tree and evaluating parent nodes for which all children have selected an action and calculated a reward bound. The best expected action for a node is selected in a manner analogous to action weight selection, except that the action with highest expected probability of having in the greatest reward is selected.

`select-action`($node$)

- for each child c of $node$ (if any), call `select-action`(c)
- set $node$'s action, $a_{node} \leftarrow argmax_a$ `prob-of-maximal-reward`($node, a$)
- call `set-reward-bounds`($node$) and return

Table 3: Algorithm for generating a plan from the BA* search tree.

The recursive function `select-action` shown in table 3, applied to the root node of the search tree will determine the best known action at each node visited. From this information a contingency plan can be generated, as before.

3.2 Discussion

In this section we have presented an efficient algorithm for finding single actor policies in MDPs with deterministic hidden states (MDPDHS). Our algorithm interleaves and old idea, search in information state space, with the new idea of optimistic/pessimistic search in state space to derive performance bounds. These bounds are used for pruning the search tree and for determining which node to expand next. Experimental comparisons will show that our approach outperforms state-of-the-art MDP and POMDP solutions to this problem.

The computation of G-values to estimate which action from a node will yield the greatest reward may seem crude at best. One limitation of the method presented is that it only compares bounds, which hide information about how each action will compare in any specific belief state (i.e., correlation between what we treat as random variables). For example, even while two actions u_1 and u_2 may have nearly indistinguishable α and β bounds, one action may *strictly dominate* the other by being preferable in all configurations of the hidden state. If this type of relationship could be determined efficiently during search, much work could be saved. An important property of the BA* algorithm, however, is that little information needs to be saved about each node in the search tree. To assert a dominance relationship between actions would require either considerable additional information about the outcomes of the actions across all possible states, or significant additional inference about the outcomes of those actions derived from the MDPDHS model at some computational expense. It is likely that in some scenarios such an approach would be beneficial, but we know of no algorithm for such inference that is generally applicable to the MDPDHS problem.

4 Coordinated Planning

The preceding section described the BA* algorithm for individual planning. When there are multiple actors and reward-providing states in the environment, the problem grows considerably in complexity. First, the existence of additional agents grows the space of states and actions since at any point an actor's decisions may be affected by the positions and actions of other actors. Second, additional goals complicate the reward function since the value of an action is affected by multiple possible future goal states. In addition, since only the first actor to reach a goal receives a reward, the presence of other actors can complicate the computation of future expected reward. Finally, if the actors can communicate, then new information can be discovered by others, and an actor's decision to make an observation may be motivated by the effect it will have on others.

In its most general form, the problem of planning for multiple communicating actors in a MDPDHS quickly becomes intractable. To deal with the inherent complexity of multiple interacting agents, we make a number of assumptions which in many circumstances allow for the generation of useful multi-actor contingency plans.

The key assumptions are:

1. **Actors can communicate.** In particular, an actor may communicate to other actors its belief state while acting, and statistics relating to its plans while it is planning.

2. **Actors plan synchronously.** All actors generate plans simultaneously and in coordination, and can periodically share information relating to those plans with each other.

3. **Actors are identical.** The actors are assumed to have identical capabilities, so that each has the same actions available in any given state and can perform those actions in the same amount of time and with the same state transition functions. Furthermore, the reward function is the same for all actors. The approach that we describe does not strictly require all of these traits, but they are assumed in the discussion and experiments.

4. **Actor observations are independent.** If one actor makes an observation which leads to a revision in its information state, the impact of the same observation on the information state of other actors will be negligible. This of course is not true, since it suggests that the agents are acting in isolated environments. However, to optimally generate plans without any such assumption requires that every actor generate contingency plans that take into account all observations that will be made by other actors. Since the actions taken by other actors are only known probabilistically in advance, this means that every observation that could possibly be made by other actors in a given information state would need to be planned for.

5. **Reward states are "consumable."** When one actor reaches a reward-providing state (referred to as a goal state in this work), it ceases to provide a reward for other actors. This condition makes

the problem more difficult, since it forces actors to coordinate to maximize a global reward function, as will be discussed below. In this work, the goal state reward function is also constrained more significantly than in the previous section.

The experiments section will demonstrate that in practice the limitations resulting from the independence assumption can be mitigated by allowing actors to replan periodically as new information becomes available through observation.

4.1 Coordinated Goal Weighting

In the discussion on single-actor planning, the method of "weighting" each node of the search tree was described. There, the weight values represented an expectation that the information state would actually be visited during execution. In this section, we develop the notion that in multi-agent planning, actors can influence each other by forming dependencies based on those weight values.

4.1.1 Goal Weighting

A difficulty in coordinated planning is that the problem is not a simple superposition of several individual planning problems. One reason for this (and the only aspect of the problem directly addressed in our approach) is that the reward function changes substantially if the goal state(s) are "consumable" and other actors may consume them first. Consider the problem of relief supply delivery to multiple locations: the reward for bringing supplies to a location diminishes over time, but disappears completely if another supply truck arrives there first. Thus any reasonable approach to planning for multiple actors must distribute agents to goals in a way that maximizes a global reward function that contains interdependecies. In the case under consideration in this project, the measured global reward is defined as:

$$R_{global} = \sum_{g \in G} e^{-\gamma T^g_{min}}$$

where G is the set of all goals, and $0 < \gamma \leq 1$ is the discount factor. T^g_{min} is the minimum time taken for an actor to reach goal g. Of particular interest is the fact that the global reward contribution from a particular goal depends only on the time at which it was first reached. As a result, the strategy for maximizing reward is to reach as many goals as quickly as possible, without needlessly sending multiple actors to the same goal state.

The first question to address is how to plan in the presence of multiple goals. The approach is straightforward: value iteration is used as before to calculate α and β bounds, but the value of a state is treated as the sum of rewards available from the individual goals. Using value iteration, the expense of this computation is approximately the same as for the single-goal case. In the presence of multiple actors we generalize this definition by introducing for each agent a a set of *goal weights*, $q^{a,g}$ that scale the expected reward seen by that agent from each goal. For agent a at state $\langle x, y \rangle$, the value of the state becomes:

$$R^a_{x,y} = \sum_{g \in G} q^{a,g} R^g_{x,y}$$

where $R^g_{x,y}$ would be the simple reward in the case of a single goal. The value of a state then becomes a weighted sum of its values with respect to each goal, where this weighting may vary between agents. In addition, we store an α and β value for each goal, such that:

$$\begin{aligned} \alpha(x, p_I) &= \sum_{g \in G} q^{a,g} \alpha^g(x, p_I) \\ \beta(x, p_I) &= \sum_{g \in G} q^{a,g} \beta^g(x, p_I) \end{aligned} \tag{7}$$

Our aim is for the actors to plan simultaneously, updating the q values in a way that represents the fractional reward expected by each actor from each goal, given the presence of the other actors in the environment.

4.1.2 Coordination by Greedy Search and Soft Goal Assignment

A very simple method of assigning goal weights q to the group of agents is the greedy method described in this section. To begin, we define a quantity which is the arithmetic mean of the α and β values at the root node of a given actor's search tree, for each goal:

$$\text{goal-reward-estimate}(g) = \frac{1}{2}(\alpha^g(x_{start}, p_{start}) + \beta^g(x_{start}, p_{start}))$$

This is a value that changes as the plan develops, since the α and β values are frequently updated during planning. It represents a very rough estimate of the reward that would be achieved from that goal alone by following the best known contingency plan yet found.

Table 4 shows the `assign-actors` procedure for setting goal weight values $q^{a,g}$ using the goal-reward-estimate defined above. The procedure is run each time that the root α or β bound of any actor changes - leading to a potential change in the goal-reward-estimate.

`assign-actors`

- while there remain unassigned actors:
  - let $\langle a, g \rangle$ be the agent-goal pair that has the highest goal-reward-estimate, where g is in the set of goals with the smallest number of actors assigned, and a has not yet been assigned.
  - set $q^{a,g} \longleftarrow z$ where $0 < z \leq 1$ is a constant
  - set $q^{a,j} \longleftarrow \frac{1-z}{|G|-1}$ for $j \neq g$, G is the set of all goals.

Table 4: Greedy procedure for assigning goal weights $q^{a,g}$ to all actors.

The constant z plays an important role in defining the "softness" of the goal assignment for actors. If $z = 1$, then it is a very hard assignment, and an actor will only "see" the goal to which it is assigned, since only that goal will influence its α and β bounds. For $z < 1$, the actor will be influenced somewhat by the presence of other goal states.

In the next section we will describe a set of experiments that measure empirically the performance of BA* in comparison to related algorithms.

5 Experimental Results

Systematic experiments were performed to evaluate the scaling properties of our algorithm in domains with hidden state. Two separate categories of experiment were performed: single actor and multiple actor.

5.1 Single Actor Experiments

We compared our algorithm to the following two algorithms: an optimized Q-learning algorithm [21], which we consider a state-of-the-art MDP solution, and Cassandra's et al. *POMDP solver software*[1] for computing policies in POMDPs [6]. Additional experimental results illustrate the effect of pruning and heuristic search using the bounds calculated via the approximate MDPs. Across the board, we found that our solution outperforms alternative approaches by a large margin. In certain experiments, our approach appears to scale linearly with the domain size, whereas the alternatives scale exponentially. We also find that the bounds are essential to keep computation low, especially in large state spaces.

The experimental world is shown in detail in Figure 5a. The problem is to move from the left end of the world to the right end. Arcs in the graph might or might not be traversable, but the only way to find out is to move there and try. The problem is to find an optimal contingency plan that enables the agent to recover if an arc is found to be impassable. This problem is known as the NP-hard *Canadian Travelers*

[1] See http://www.cassandra.org/pomdp/code/

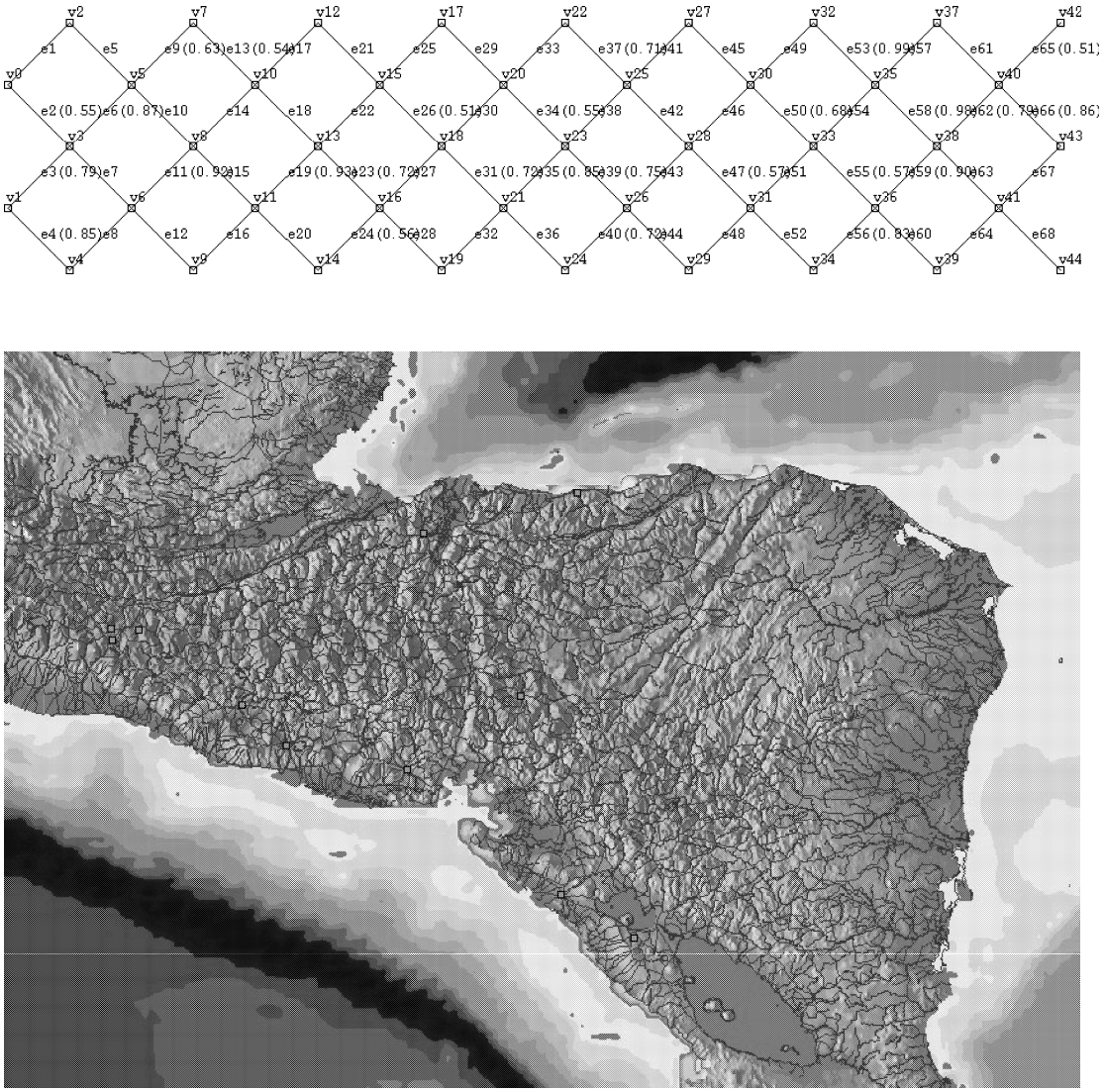


Figure 5: (top) Example test graph for systematic experiments. (bottom) Road map of Honduras, used for large-scale experiments.

Problem [16], and is very similar to the mobile robot exploration problem [19]. The advantage of an artificial environment such as the one used here is that it complexity can easily be varied, by adding new nodes to the graph. The number of arcs, and hence the number of hidden states, is linear in the number of nodes in the graph. As a direct consequence, the belief state space is exponential in the size of the state space.

The "rules" of the environments are:

1. The cost of travelling on an arc is proportional to its cartesian length as pictured.
2. The cost of discovering that an arc is not traversable is the same as travelling its length.
3. The hidden state in the environment is randomized with each simulated run, in accordance with the actor's initial information state probabilities. Thus, the actor's model of the environment is completely accurate.
4. There exists a single "goal" state, which is the only reward-providing state in the environment. Upon being reached, the simulation terminates.

The *POMDP solver* software was only able to generate policies in extremely small worlds. This is not surprising, as POMDPs are more general than MDPDHSs. The timing results are as follows, broken down into the time required for finding an initial policy and for finding the optimal policy:

number of nodes in graph	initial solution	optimal solution
4	5 sec	$\sim$5,000 sec
5	8 sec	$\sim$7,000 sec
8	$>$ 3,000 sec	$>$ 24hrs

We therefore did not investigate this approach any further.

To compute α and β bounds for BA* in the CTP, it is necessary to find the information states that result in minimal and maximal expected reward. The maximal reward is easily found by assuming that all unknown-state edges are traversable. Finding the minimal reward information state is somewhat more complicated, since the worst case is a long path that explores many untraversable edges (since the actor may not infer that it is in the worst case environment until it has done quite a bit of exploration.) In the experiments described in this section, we approximate the lower α bound by deriving the shortest-path distance to the goal given that the actor knows that all hidden-state edges are not traversable. It is not unlikely that this approach may in some cases result in improper pruning of the search tree, and suboptimal ordering of state expansions. However, since all actions being compared are subject to the same method of lower bound approximation, the effect on action *preference* is not expected to be large.

Due to the restrictive nature of MDPDHSs, the MDP solution scales much better. As stated above, Q-learning was used on the discrete information state space. Figure 6a compares the performance of Q-learning (dashed line) and our search algorithm (solid line) under equal computational conditions. The figure shows the probability of reaching the goal using a policy calculated with fixed computational time, and it can be seen that BA* can handle significantly larger worlds. Notice that the horizontal axis depicts the size of the graph. The size of the information space is much larger. The Q-learning algorithm faces two primary disadvantages in the CTP environment. First, it does not use a model of transition probabilities, using instead an inefficient method of learning a mapping from state-action pairs to expected reward. Even more problematic, however, is that the mapping is from individual states to actions. As a result, a large portion of the information state must be represented internally to develop a plan.

The results presented so far reflect the performance if the policy is calculated up front and never revised as the agent moves. A common strategy for boosting the performance in worlds with hidden state is to replan whenever new information arrives. Figures 6c&d show performance results for such a setting. In particular, Figure 6c shows the average path length of the controller identified by Q-learning (dashed curve) and compares it with the average path length obtained using our heuristic search algorithm. Notice that our approach scales significantly better. From worlds with 45 states on, Q-learning fails to find any policy that even contains a single path to the goal in reasonable time, despite replanning. This is because the initial planning step never finds even a single path to the goal. Our approach navigates successfully in much larger worlds. Figure 6d shows related results: Plotted here is the computation time required to achieve a fixed solution quality. Again, the Q-learning implementation scales much poorer to complex worlds than our new heuristic search algorithm.

These results raise the question as to *why* the search algorithm is so much better than Q-learning. We conjecture that our approach for generating bounds and using them to prune and order the search makes the difference. Figure 6b illustrates the effect of the bounds by plotting the number information states expanded in the search, for our algorithm and for breadth-first search (which is our algorithm deprived of the bounds). Notice that the vertical axis uses a logarithmic scale. Obviously, our bounds reduce the number of nodes expanded by more than three orders of magnitudes, and the gap increases with the size of the world. We view this as a key result for motivating the MDPDHS model—instead of subsuming the type problems addressed in this paper into the more common MDP model. It is the specific hidden state structure that makes our pruning and heuristic search methods possible.

We also applied our algorithm to a large-scale navigation problem using data of Honduras released by the USGS (see Figure 5b), as part of a DARPA-sponsored research program on disaster relief. The specific scenario involved roads randomly destroyed by Hurricane Mitch, which created a fancy version of the Canadian Travelers Problem. Our map contained 234 nodes and 300 major road segments, creating an information state space of size $4.7 \cdot 10^{90}$ states. Using replanning, our approach consistently

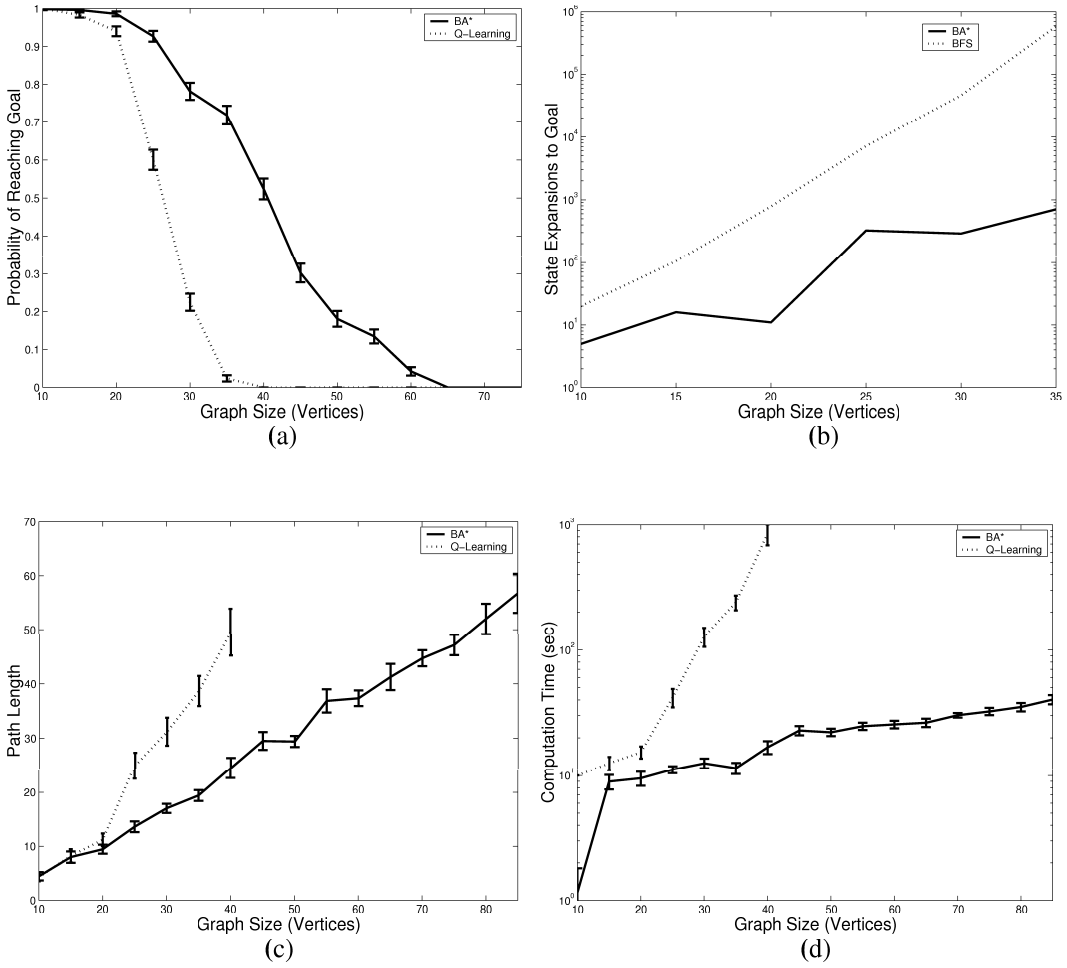


Figure 6: Performance curves, comparing Q-learning with explicit encoding of the hidden state with our new search algorithm. In all graphs, the horizontal axis depicts the number of nodes in the world. (a) Success rate for a single policy calculated up front. (b) Number of nodes expanded by our approach versus breadth first search. (c) Average length of the path to the goal, using replanning. (d) Average computation time required to find a policy of fixed quality.

managed to move from any location to any other on a convincingly short path, assuming that such a path exists in the first place. We are not aware of any other algorithm that could handle domains of this size, but we also cannot assess how close to optimal our approach performs.

It should be noted that in all but the smallest environments illustated here, BA* does not find the optimal algorithm in time and memory allotted to it, so the results are based largely on the performance of BA* when used as an any-time algorithm with less than a minute of computation time available per actor on a modern computer.

5.2 Multiple Actor Experiments

To evaluate the effectiveness of our approach in domains with hidden state and multiple actors and goal states, we revisit the large-scale experimental domain of the previous section. In this second set of experiments, we investigate the scaling properties of the algorithm as the number of actors increases, and compare to a method that uses the BA* algorithm at the individual level, but does not dynamically reweight goal assignments as described in section 4.1.2.

We do not know of other multiple actor planning techniques suited to the CTP domain. A simple alternative to the method presented here would be to use an approach based solely on value iteration, but

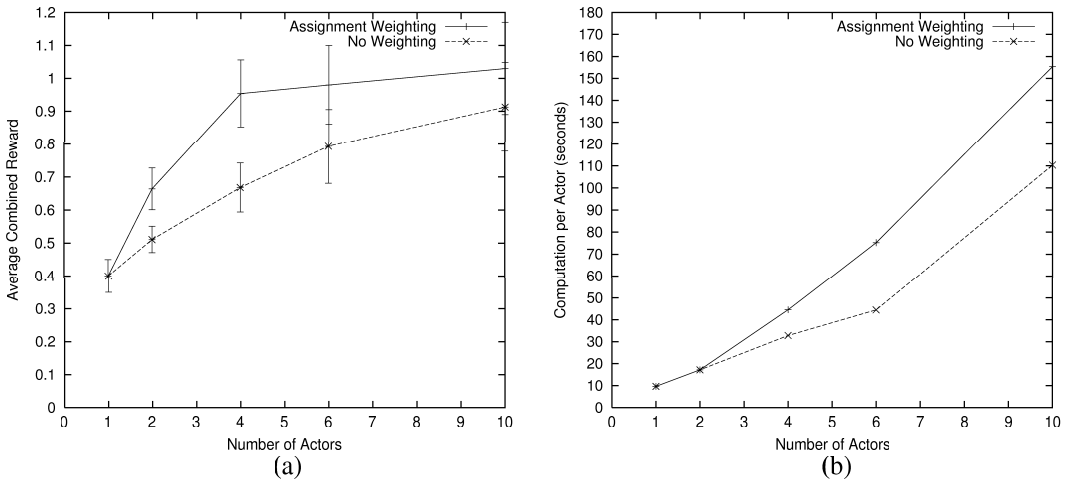


Figure 7: A comparison of (a) reward and (b) computation time, for multi-actor BA* with and without dynamic goal weighting using greedy assignment.

we have already shown that this variety of algorithm is deficient in the single-actor case, and that it does not scale well to larger environments, particularly those that result from multiple interacting agents.

The "rules" of the multi-actor simulation are the same as in previous experiments, with the following additional points:

1. There are 4 goal states, and the number of actors is varied between 1 and 10.
2. As soon as an actor reaches one goal state, it may travel to another and reap additional reward there.
3. In all environments, every goal is potentially reachable by at least one actor. Randomly selected environments that failed this criteria were skipped in order to reduce the variance of the resulting rewards. This has skewed the distribution of edge states slightly from that modeled by the actors, but not to a degree that is at all likely to be significant.

Figure 7 compares the performance of multi-actor BA* with and without greedy weighting. As should be expected, using more agents increases the reward acheived. The increase in computation by both methods stems from the fact that each time an agent makes an observation that changes its distribution p_I, all agents replan immediately. Thus, with more agents, there are more replanning rounds so more computation is required.

Nonetheless, that there is more overhead in the case where goal weighting is used is reflected in the increasing disparity in planning time required as the number of actors increases. To verify that the BA* goal weighting algorithm can handle many actors, an informal experiment was performed with 5 goals and 1,500 agents. The resulting computation was dominated by actor node-expansion, and not the goal weighting step.

Because the two approaches behave identically when there is a single actor, the performance is expected to be the same at the first data point. However, as the number of actors increases, the greedy weighting approach distributes actors more evenly to the goals that they are best capable of reaching, resulting in significantly higher rewards. The main reason for the oberved difference in global reward is that goal weighting mechanism resolves contention between multiple actors that expect to maximize their local reward by moving to the same goal, irrespective of the effect of the other agent on that future reward.

As the number of actors reaches and exceeds the number of goals, the increase in performance by both methods starts to level off, and the lead enjoyed by the goal weighting approach begins to diminish. Reward gains made by better allocation of a sparse group of agents are overwhelmed by the ability to task a large number of agents when they are available.

The results shown above demonstrate that the greedy goal weighting approach to BA* planning is an improvement over the unweighted technique in a simulated environment.

6 Related Work

Little published research has been applied to planning in the Canadian Traveler's Problem (CTP) domain, which is the case addressed in our experiments. The single-agent BA* algorithm [9] 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.

Reinforcement learning has often been applied in environments similar to MDPDHSs [22]. One approach has been to ignore the hidden state, treating it as part of the uncertainty in a MDP, but this throws away information obtainable through observations, limiting the quality of the solution. To effectively solve the problem in the reinforcement learning framework, it becomes necessary to explicitly represent the entire space of information states, which grow exponentially with the size of the hidden state. As shown in the experimental results section, the Q-learning reinforcement learning algorithm is hindered by lack of a pre-defined model of the environment and by the need to explicitly represent many information states. Even reinforcement learning algorithms that can make use of prior knowledge of the environment must store exponentially more state information since there is no mechanism for pruning according to reward expectation.

The CTP can be cast as a very special case of a Partially Observable Markov Decision Process (POMDP), an area that has received much 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[6, 10]. 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)[17], as was confirmed in the experiments section. 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 BA* approach is bounded search in game-playing. The $\alpha\beta$ algorithm is the prime example [7], but numerous stochastic extensions have been proposed [4, 15]. 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. [2] 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.

7 Conclusions

As stated in the introduction to this paper, we believe that MDPs and POMDPs are only the two ends of a spectrum of sequential decision-making problems under uncertainty. We hope that this work moti-

vates future research on extensions of MDPs that capture some relevant aspects of POMDPs, yet lend themselves to computationally more efficient solutions than the full-blown POMDP solution.

The problem of planning for multiple actors in MDPDHSs is clearly still open, but the BA* framework presents a novel framework for coordinating several goal-seeking contingency plans. This project has examined one method for using the information provided by the planning algorithm to achieve improved results within a set of problem domains. In addition, further research exploring potential dominance relationships between actions when computing action preferences could yield benefits in a number of domains.

8 Acknowledgments

Thank you to Sebastian Thrun, a great advisor and friend, for many encouraging and helpful discussions, and for bringing about the formal MDPDHS model. Thanks to Lucian-Vlad Lita for many valuable conversations and help with experiments. This work was funded by DARPA under the Control of Agent Based Systems program.

References

[1] M. Bardi, Parthasarathym T., and T.E.S. Raghavan. *Stochastic and Differential Games: Theory and Numerical Methods*. Birkhauser Verlag AG, 1999.

[2] Eric B. Baum and Warren D. Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1–2):195–242, 1997.

[3] J. Baxter, L. Weaver, and P. Bartlett. Experiments with infinite-horizon, policy-gradient estimation. *JAIR*, 15:351–381, 2001.

[4] M. Buro. ProbCut: An effective selective extension of the alpha-beta algorithm. *International Computer Chess Association Journal*, 18(2):71–76, 1995.

[5] A.M. Jazwinsky. *Stochastic Processes and Filtering Theory*. Academic, 1970.

[6] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[7] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[8] S. Koenig. The complexity of real-time search. Technical Report CMU-CS-92-145, Carnegie Melloon University, 1992.

[9] L.V. Lita, J. Schulte, and S. Thrun. A multi-agent system for agent coordination in uncertain environments. In *Proceedings of the 5th International Conference on Autonomous Agen ts*, 2001.

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

[11] Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI–95)*, pages 394–402, Montreal, Québec, Canada, 1995.

[12] Donald Michie. Memo functions and machine learning. *Nature*, 218(1):19–22, April 1968.

[13] A.Y. Ng and M. Jordan. PEGASUS: a policy search method for large MDPs and POMDPs. In *UAI*, 2000.

[14] N. Nilsson. *Principles of Artificial Intelligence*. Springer, 1982.

[15] A. J. Palay. *Searching with probabilities*. Pitman Publishing, 1985.

[16] C. Papadimitriou and M. Yannakakis. Shortest paths without a map. In *ICALP*, 1989.

[17] R. Parr and S. Russell. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[18] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[19] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *AAAI*, 2000.

[20] E. Sondik. *The Optimal Control of Partially Observable Markov Processes*. PhD thesis, Stanford University, 1971.

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

[22] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[23] R. J. Williams and L. C. Baird III. Analysis of some incremental variant of policy iteration: First steps toward understanding actor-critic learning systems. Technical Report NU-CCS-93-11, Northeastern University, 1993.