ASU CSE TR-07-006; June 2007

Probabilistic Planning *is* Multi-objective!

Daniel Bryce
Artificial Intelligence Center, SRI International, Inc.
333 Ravenswood Ave, Menlo Park, CA 94025
bryce@ai.sri.com

William Cushing, & Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University
699 South Mill #501 Avenue, Tempe, AZ 85281
{wcushing, rao}@asu.edu

June 13, 2007

Abstract

Probabilistic planning is an inherently multi-objective problem where plans must trade-off probability of goal satisfaction with expected plan cost. To date, probabilistic plan synthesis algorithms have focussed on single objective formulations that bound one of the objectives by making some unnatural assumptions. We show that a multi-objective formulation is not only needed, but also enables us to (i) generate Pareto sets of plans, (ii) use recent advances in probabilistic planning reachability heuristics, and (iii) elegantly solve limited contingency planning problems. We extend LAO^* to its multi-objective counterpart $MOLAO^*$, and discuss a number of speed-up techniques that form the basis for a state of the art conditional probabilistic planner.

1 Introduction

Probabilistic planning is the inherently multi-objective problem of optimizing both plan cost and probability of goal satisfaction. However, as both Kushmerick, Hanks, & Weld (1994) and Littman (1997) point out, it is possible to bound one of the objectives and optimize the other with a single objective search. For example, conformant (non-observable) probabilistic planning can be formulated as a forward-chaining (single objective) A* search in the space of belief states that optimizes plan cost. If the search finds a belief state satisfying the goal with probability no less than τ, then the action sequence leading to it is a feasible plan.

When it comes to conditional planning however, the situation is more complex. The probability of satisfying the goal is the expectation of satisfying it in each branch. To ensure plan feasibility (i.e., the probability of goal satisfaction exceeds τ), we must consider both (i) the probability of executing each branch and (ii) the probability that each satisfies the goal. In conformant planning (i) is always 1.0, making (ii) a belief state property. In conditional planning, feasibility is a property of the entire plan (through aggregate probabilities of branches). Naturally, during synthesis there may be many sub-plan options within the different branches. Finding a cost-optimal feasible plan involves deciding between sub-plans (in each branch) that have different costs and probabilities of goal satisfaction. Unfortunately, as the following example illustrates, it is not possible to decide which sub-plan is best in one plan branch without considering sub-plan options in the other branches. That is, we need to somehow combine these *non-dominated* options across branches.

Example: *The conditional plan π (Figure 1) starts with action a, which provides two observations and has an execution cost of one. In the branch corresponding to the first observation (whose probability is 0.2) it is possible to execute sub-plan π_1 that achieves the goal with 1.0 probability and expected cost 50 or π_2 with 0.5 probability and expected cost 10. In the branch for the second observation (whose probability is 0.8) it is possible to execute sub-plan π_3 that achieves the goal with 0.75 probability and expected cost 30 or π_4 with 0.0 probability and expected cost 0. The sub-plan choices result in different plans:*

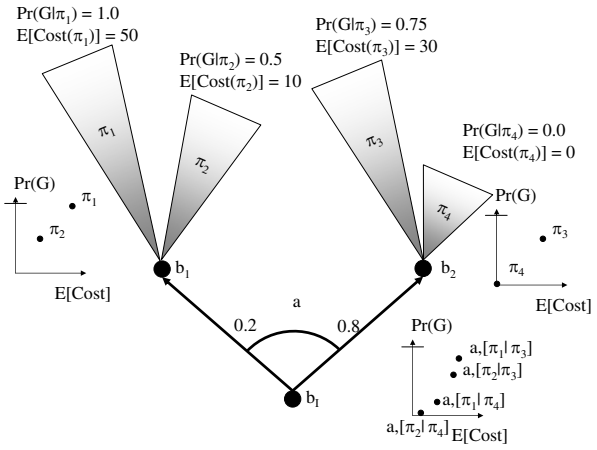

Figure 1: Example of Conditional Probabilistic Planning.

Plans π	E[Cost(π)]	Pr($G|\pi$)
$a, [\pi_1 \vert \pi_3]$	1+(.2)50+(.8)30 = 35	(.2)1+(.8).75 = .8
$a, [\pi_2 \vert \pi_3]$	1+(.2)10+(.8)30 = 27	(.2).5+(.8).75 = .7
$a, [\pi_1 \vert \pi_4]$	1+(.2)50+(.8)0 = 11	(.2)1+(.8)0 = .2
$a, [\pi_2 \vert \pi_4]$	1+(.2)10+(.8)0 = 3	(.2).5+(.8)0 = .1

If $\tau = 0.6$, then using sub-plans π_2 and π_3 exceeds the threshold with a minimal expected cost of 27. Notice that if we commit to sub-plans in each branch without considering other branches, we may choose π_1 and π_3 (because each individually exceeds τ). Yet, this aggressive (and incomplete) strategy finds a feasible plan with a *higher* cost of 35.

Our first contribution (a solution to this problem), is to delay such pre-mature commitments in each branch with bottom-up dynamic programming in belief state space. Each belief state maintains a set of non-dominated sub-plans and, through multi-objective dynamic programming (Henig, 1983), communicates these options to its ancestors. It is only possible at the root (initial belief state) to select the sub-plans that correspond to a cost-optimal feasible plan.

Since we cannot enumerate the (infinite sized) belief state space to perform multi-objective value iteration, our second contribution is to search forward from the initial belief state using a multi-objective generalization of the LAO^* algorithm, called $MOLAO^*$. Since the cost-optimal feasible plan is one of several non-dominated plans, $MOLAO^*$ readily computes Pareto sets of plans.

At first glance, it would seem as if combining multi-objective optimization and probabilistic planning (two notoriously computationally difficult tasks) will not scale. Our third contribution is aimed at taming the complexity of the search, by pursuing a number of speed-ups. The most notable of these speed-ups is to use existing planning graph reachability heuristics (Bryce, Kambhampati, & Smith, 2006) to guide the search. We also discuss variations on $MOLAO^*$ that focus synthesis on likely branches, reduce the complexity of dynamic programming, and reduce the size of Pareto sets by using ϵ-domination (Papadimitriou & Yannakakis, 2003).

We also take advantage of the multi-objective formulation to include other objectives. In our fourth contribution, we pursue limited contingency planning (LCP). Our approach to LCP trades the complexity of increasing the state space, as Meuleau & Smith (2003), for the complexity of using another objective: the number of plan branches.

In the next section, we formally define our planning model. The following section discusses two single objective dynamic programming formulations of conditional probabilistic planning and shows how they rely on very restrictive assumptions. We then show how a *novel* multi-objective formulation lifts these assumptions in the successive section. The next section describes $MOLAO^*$ and our speed-ups, followed by an empirical evaluation, a discussion of limited contingency planning, related work, and conclusions.

2 Background & Representation

We describe our planning models in terms of flat Markov decision processes (MDPs), but note that our underlying implementation is propositional. We are interested in partially observable planning, but to simplify later sections, we describe the partially observable model as an equivalent fully-observable belief state MDP.

Full Observability: The fully-observable model is given by (S, A, T, s_I, G), defined as a set of states S, a set of actions A, a state transition relation $T(s, a, s')$, an initial state s_I, and a goal state function $G(s)$. The state transition relation describes a probability distribution: $T(s, a, s') = Pr(s'|a, s)$. A subset of actions $A(s) \subseteq A \cup \perp$ is applicable in each state s. The $\perp$ action signifies no action is performed in s. The goal state function $G : S \rightarrow [0, 1]$ maps each state to a probability it satisfies the goal. In this model, $G(s) = 1$ or $G(s) = 0$ for every state (later we will see that $0 \leq G(s) \leq 1$ in belief state MDPs). Applying an action incurs a cost $c(a)$, which we will assume is uniform in this paper (with the exception that $c(\perp) = 0$).

Partial Observability: The partially-observable MDP model is given by $(S, A, T, b_I, G, O, \Omega)$, defined as before, and by an initial belief state b_I, an observation function $O(s, a, o)$, and a set of observations Ω. A belief state $b : S \rightarrow [0, 1]$ is a probability distribution that maps each state to a probability, such that $\sum_{s \in S} b(s) = 1.0$. We say that $s \in b$ for all states where $b(s) \geq 0$. The set $A(b) = \cap_{s \in b} A(s)$ contains all actions applicable in a belief state b. The observation function $O(s, a, o) = Pr(o|a, s)$ is the probability distribution over observations $o \in \Omega$ received upon transitioning to state s after executing action a.

We define the belief state b_a reached by applying a in belief state b as $b_a(s') = \sum_{s \in S} b(s) T(s, a, s')$. The belief state b_a^o is the belief state after receiving observation o in belief state b_a, defined as $b_a^o(s') = \alpha O(s', a, o) b_a(s')$, where α is a normalization factor. The probability of receiving observation o in belief state b_a is $T(b, a, o, b_a^o) = \alpha = \sum_{s \in S} b_a(s) O(s, a, o)$.

Belief state MDPs: To clarify the correspondence between the fully-observable and partially observable model, consider the following transformation, called a belief state MDP. The belief state MDP $(\tilde{S}, A, \tilde{T}, \tilde{s}_{b_I}, \tilde{G})$ for the partially-observable MDP $(S, A, T, b_I, G, O, \Omega)$ is given by several equivalences, where each state $\tilde{s}_b \in \tilde{S}$ corresponds to a belief state b, and $\tilde{s}_{b_I}$ is the initial (belief) state. A is an unchanged set of actions, where $A(\tilde{s}_b) = A(b)$ is the set of actions applicable in b. The transition relation $\tilde{T}(\tilde{s}_b, a, \tilde{s}_{b_a^o})$ is equivalent to $T(b, a, o, b_a^o)$. $\tilde{G}$ is the probability each (belief) state satisfies the goal, such that $\tilde{G}(\tilde{s}_b) = \sum_{s \in S} b(s) G(s)$. In the following, we ignore all distinctions between belief state MDPs and fully-observable MDPs, where possible, given the above equivalences.

3 Single Objective Formulations

Most works that address conditional probabilistic planning optimize a single objective: probability of goal satisfaction or expected cost. We describe these approaches and characterize their Bellman optimality equations.

Maximizing Probability: Optimizing the probability of goal satisfaction completely ignores plan cost, except, perhaps, when breaking ties between plans that satisfy the goal with the same probability. With no concern for plan cost, it is possible to search for increasingly longer plans to improve the probability of goal satisfaction. An often used solution to this problem is to assume a finite horizon (Majercik & Littman, 2003; Little, Aberdeen, & Theibaux, 2005), and find the highest probability k horizon plan. The Bellman equation to minimize the probability of not satisfying the goal for each state s, for each horizon $0 \leq t < k$ is:

$$\begin{aligned} J(s,t) &= \min_{a \in A(s)} q(s,t,a) \\ \pi(s,t) &= \operatorname*{argmin}_{a \in A(s)} q(s,t,a) \\ q(s,t,a) &= \sum_{s' \in S} T(s,a,s') J(s', t+1) \end{aligned}$$

and $J(s, k) = 1 - G(s)$. The objective is to compute the minimal $J(s_I, 0)$ and the corresponding plan π.

In the example (Figure 1), each sub-plan π_1 to π_4 could have the same expected length, assuming there are persistence actions. If $k = 51$, then the optimal plan would select sub-plans π_1 and π_3 because they maximize the probability of goal satisfaction. This formulation does not explicitly permit heterogenous length plan branches, nor does it accommodate action costs.

Minimizing Expected Cost: Another solution is to optimize expected cost to reach the goal. It is customary to assume a cost $c_{\not G}$ of not reaching the goal to make the objective a linear combination of probability and cost. This is analogous

to reward models that assign negative reward to actions and positive reward to goal states. The Bellman optimality equation for minimizing expected cost for each state s is:

$$\begin{aligned} J(s) &= \min_{a \in A(s)} q(s,a) \\ \pi(s) &= \operatorname*{argmin}_{a \in A(s)} q(s,a) \\ q(s,a) &= c(a) + \sum_{s' \in S} T(s,a,s') J(s') \\ q(s,\perp) &= c_{\not G}(1 - G(s)) \end{aligned}$$

The objective is to compute the minimal $J(s_I)$ and the corresponding plan π.

Each choice for $c_{\not G}$ indirectly influences the optimal probability of satisfying the goal, given the costs of potential plans. To see this, consider how the value for $c_{\not G}$ affects the choice between π_1 and π_2 in the example. If $c_{\not G}$ = 79, then π_1 has a total cost of (1-1)79+50 = 50 and π_2 has a total cost of (1-0.5)79+10 = 49.5, making π_2 a better choice. If $c_{\not G}$ = 81, then π_1 has a total cost of (1-1)79+50 = 50 and π_2 has a total cost of (1-1)81+10 = 50.5, making π_1 a better choice. However, if $c_{\not G}$ = 80, then π_1 has a total cost of 50 and π_2 has a total cost of 50, making the choice ambiguous. With a small variation in $c_{\not G}$ the best plan (and the probability of goal satisfaction) can change. To model a problem with a probability of goal satisfaction threshold requires knowledge of the costs of alternative plans. However, if we knew the costs of alternative plans, then what is the point of using an automated planner?! Nevertheless, optimizing expected cost/reward does have its merits when the probability of goal satisfaction is not of primary concern to the domain modeler.

4 Multi-objective Formulation

Instead of carefully engineering problems by imposing a horizon bound or a cost for failing to satisfy the goal, it is possible to directly solve the conditional probabilistic planning problem with a more natural multi-objective formulation. The formulation can (i) minimize plan cost, subject to a lower bound on the probability of goal satisfaction (omitting a horizon bound and goal cost), or (ii) find a Pareto set of plans when there is no bound on the probability of goal satisfaction. Committing to a multi-objective formulation opens the possibility for other objectives (e.g., number of branches, makespan, resource consumption, etc.), so we present a general multi-objective formulation and point out specific details for computing the expected cost and probability of goal satisfaction objectives. In a later section, we discuss limited contingency planning by introducing the number of branches as a third objective.

We introduce an extension to the value function, such that $J(s)$ becomes a Pareto set. Each point of the Pareto set $q(s,a) \in J(s)$ represents a vector of n objectives, such that $q(s,a) = (q_0(s,a), q_1(s,a), ..., q_{n-1}(s,a))$, and $J(s) = \{q(s,a) | \neg \exists_{q'(s,a') \in J(s)} q'(s,a') \prec q(s,a)\}$
where $J(s)$ contains only non-dominated points. A point $q'(s,a')$ dominates another $q(s,a)$ if it is as good in all objectives, and strictly better in at least one:

$$q'(s,a') \prec q(s,a) \Leftrightarrow \begin{array}{l} \forall_i q'_i(s,a') \leq q_i(s,a) \wedge \\ \exists_i q'_i(s,a') < q_i(s,a) \end{array}$$

Each point is mapped to a best action by π (i.e., $\pi(s, q(s,a)) = a$), making it possible for two sub-plans to start with the same action but have different values.

Each $q(s,a) \in J(s)$ is computed in terms of its successors' values. Because each successor ($s', s'', ...$) is associated with a Pareto set ($J(s'), J(s''), ...$), it is possible to define a different $q(s,a)$ from each element ($q(s',a'), q(s'',a''), ...$) of the cross product of the Pareto sets ($J(s') \times J(s'') \times ...$). We saw this in the example, where it was possible to have $|J(\tilde{s}_{b_I})| = 4$ because $|J(\tilde{s}_{b_1})| = |J(\tilde{s}_{b_2})| = 2$. The cross product of $J(\tilde{s}_{b_1})$ and $J(\tilde{s}_{b_2})$ contains four elements, each used to define an element of $J(\tilde{s}_{b_I})$. The actual number of elements in a Pareto set may be less because some elements will be dominated.

We define expected cost and probability of *not* satisfying the goal objectives for probabilistic conditional planning, $q(s,a) = (q_0(s,a), q_1(s,a))$, such that:

$$\begin{aligned} q_0(s,a) &= c(a) + \sum_{s' \in S} T(s,a,s') q'_0(s',a') \\ q_0(s,\perp) &= 0 \\ q_1(s,a) &= \sum_{s' \in S} T(s,a,s') q'_1(s',a') \\ q_1(s,\perp) &= 1 - G(s) \end{aligned} \tag{1}$$

Each $q'(s',a')$ in the above equations is from an element of the cross product of successor Pareto sets.

From the example, $J(\tilde{s}_{b_I}) = \{(35, 0.2), (27, 0.3), (11, 0.8), (3, 0.9)\}$. The first element of $J(\tilde{s}_{b_I})$ is defined by $q(\tilde{s}_{b_1}, (50, 0.0)) \in J(\tilde{s}_{b_1})$ and $q(\tilde{s}_{b_2}, (30, 0.25)) \in J(\tilde{s}_{b_2})$ as $q_0(\tilde{s}_{b_I}, a) = 1 + (0.2)50 + (0.8)30 = 35$ and $q_1(\tilde{s}_{b_I}, a) = (0.2)0.0 + (0.8)0.25 = 0.2$.

To solve the probabilistic conditional planning problem, we are interested in finding a plan starting with the action $a = \pi(s_I, (q_0(s_I, a), q_1(s_I, a)))$ such that the plan has minimal cost $q(s_I, a) = \min_{q(s_I,a) \in J(s_I)} q_0(s_I, a)$ subject to the goal satisfaction threshold $\tau \leq 1 - q_1(s_I, a)$. The remaining actions in the plan are those actions used to define each $q(s, a)$ for the actions in the plan. For example, if the plan starts by executing a and it results in a single state s', then the action to execute in s' is a' where $q'(s', a')$ was used to compute $q(s_I, a)$.

Computing the Multi-objective Bellman Equation: Computing the multi-objective Bellman equation *once* for a state s can take $O(|A(s)||J(s')|^{|S|})$ time because $|A(s)|$ actions are applicable in s, each results in at most $|S|$ successor states s', and there are $|J(s')|$ Pareto optimal sub-plans for each state s'. In a later section, we identify ways to reduce this complexity by limiting the size of the Pareto sets $J(s)$.

5 Multi-objective LAO^*

Hansen & Zilberstein (2001) introduce Looping AO* (LAO^*) search as a technique for solving stochastic shortest path and MDP problems. Unlike the traditional value iteration and policy iteration techniques for solving MDP problems, LAO^* generalizes AO* search (Nilsson, 1980) to handle loops in the search graph. The idea is to expand a limited region of the state space, over which value iteration is used to identify a partial plan. In each iteration, LAO^* expands unexpanded fringe states of the partial plan, and performs value iteration. If the fringe state values are initialized with an admissible heuristic, LAO^* terminates when it identifies an optimal solution. LAO^* termination relies on two criterion: the difference between upper and lower bounds on the value function is below a threshold, and the solution contains no unexpanded states. In our setting, solutions are plans and not policies, meaning every plan path eventually ends in a terminal state. By labeling terminal states as "solved" (as done in AO*), we can propagate solved labels to identify if a solution contains unexpanded states.

The main generalization needed for a multi-objective LAO^* ($MOLAO^*$) is to reinterpret the J-values as J-sets and compute them as such. This also means that there may be several non-dominated partial solutions that $MOLAO^*$ can expand each iteration. Figures 2 and 3 describe the $MOLAO^*$ search algorithm using J-sets to find a Pareto set of conditional plans. The $MOLAO^*$ function in Figure 2 contains the outer loop that calls `ExpandPlan` to compute a set of states Z that are reachable by the set of non-dominated partial plans. To Z, `AddAncestors` adds all states that can reach a state $s \in$ Z with a non-dominated plan. The set of partial plans is revised by calling `VI`(Z) to update $J(s)$ for each $s \in$ Z. When `ConvergenceTest` indicates that $J(s_I)$ has converged (i.e, all non-dominated solutions are labeled solved, and upper and lower bounds on their values is below a threshold), it is possible to stop.

The `ExpandPlan` function recursively traces each non-dominated partial plan (lines 7-13) to find leaf states to expand (lines 4-5). The `ExpandState` function applies each action $a \in A(s)$ to generate successor states. Each successor state s' has its Pareto set $J(s')$ initialized and expanded(s') is set equal to zero. Initializing $J(s')$ involves adding a point $q(s', \perp)$ where solved($q(s', \perp)$) is true, indicating that it is possible to end the plan at s'. We can also add heuristic points $q(s, *)$ to $J(s)$ that indicate heuristic estimates of non-dominated sub-plans. Each heuristic point is marked unsolved. Through dynamic programming, each $J(s)$ will contain some heuristic points that indicate the value of partial plans rooted at s. Later, we discuss which and how many heuristic points we use.

The `VI` function performs value iteration on a set of states (i.e., iteratively calling `Backup`, Figure 3, for each state until the maximum change in some $J(s)$ falls below a threshold). The `Backup` function computes $J(s)$ by applying every action in the loop (lines 2-9). For each action, W is the cross product of Pareto sets of successor states (line 3). For each element $w \in W$, a point $q(s, a)$ is computed (as in Equation 1) with the points $q'(s', a') \in w$ (line 5). Each new point is marked solved if each of the successor points is solved (line 6). The `UpdateDominance` function (line 7) maintains the Pareto set by checking if each new point is dominated, or dominates points already in the Pareto set.

We deliberatively leave `ConvergenceTest` and `ComputeError` undefined because space precludes a thorough analysis of the solution error. Instead, in the following, we introduce a variation on $MOLAO^*$ that we use to compute *feasible* conditional probabilistic plans quickly. Many of the speed-ups involved will affect completeness and admissibility. In future work, we will analyze the value iteration error and conditions for optimality by using the notion of hyperarea difference (Wu & Azram, 2001).

```
MOLAO*()
 1: i = 0
 2: repeat
 3:    Z = ExpandPlan(s_I, i++)
 4:    AddAncestors(Z)
 5:    VI(Z)
 6: until (ConvergenceTest(s_I) == T)

ExpandPlan(s, i)
 1: if expanded(s) < i then
 2:    Z = Z ∪ s
 3:    if expanded(s) == 0 then
 4:       expanded(s) = i
 5:       ExpandState(s)
 6:    else
 7:       expanded(s) = i
 8:       for q(s,a) ∈ J(s) s.t. ¬solved(q(s,a)) do
 9:          for s' ∈ S : T(s,a,s') > 0 do
10:             Z' = ExpandPlan(s', i)
11:             Z = Z ∪ Z'
12:          end for
13:       end for
14:    end if
15: end if
16: return Z
```

Figure 2: $MOLAO^*$ Search Algorithm.

```
Backup(s)
 1: J'(s) = ∅
 2: for a ∈ A(s) do
 3:    W = ×_{s':T(s,a,s')>0} J(s')
 4:    for w ∈ W do
 5:       Compute q(s,a) with each q'(s',a') ∈ w
 6:       solved(q(s,a)) = ⋀_{q'(s',a')∈w} solved(q'(s',a'))
 7:       UpdateDominance(q(s,a), J'(s))
 8:    end for
 9: end for
10: error = ComputeError(J'(s), J(s))
11: J(s) = J'(s)
12: return error
```

Figure 3: $MOLAO^*$ state value `Backup`.

5.1 $MOLAO^*$ Speedups

In the following, we describe four improvements to $MOLAO^*$ that help find feasible, but suboptimal, solutions quickly. The first is a modified version of $MOLAO^*$, named m$MOLAO^*$. The second describes heuristics. The third involves Pareto set approximations. The fourth simulates the current partial plan to focus synthesis, similar to RTDP (Barto, Bradtke, & Singh, 1995).

m$MOLAO^*$**:** The variation of $MOLAO^*$ that we pursue is very similar to the "efficient" version of LAO^* presented by Hansen & Zilberstein (2001) (c.f. Table 7). The idea is to combine value iteration with solution expansion by

```
mMOLAO*()
 1: i = 0
 2: repeat
 3:    mExpandPlan(s_I, i++)
 4: until ∃_{q(s_I,a)∈J(s_I)} feasible(q(s_I, a))

mExpandPlan(s, i)
 1: if expanded(s) < i then
 2:    if expanded(s) == 0 then
 3:       expanded(s) = i
 4:       ExpandState(s)
 5:    else
 6:       expanded(s) = i
 7:       for q(s, a) ∈ J(s) s.t. ¬solved(q(s, a)) do
 8:          for s' ∈ S : T(s, a, s') > 0 do
 9:             mExpandPlan(s', i)
10:          end for
11:       end for
12:    end if
13:    Backup(s)
14: end if
```

Figure 4: Modified $MOLAO^*$ Search Algorithm.

calling the `Backup` function for each state after recursively calling `ExpandPlan` for each of child state reached by an action in $J(s)$. Figure 4 describes our version of the $MOLAO^*$ algorithm that not only combines value iteration with solution expansion, but also stops when a *feasible* (labeled solved) solution satisfies the goal with probability no less than τ. We omit the convergence test used in LAO*, so that we can stop at the first feasible plan.

$MOLAO^*$ **Heuristics:** Adding heuristic points increases the size of J-sets, which we previously noted as a major source of complexity in computing state backups. There are two techniques we use to mitigate the increase in J-set size. First, we use a single heuristic point that estimates the cost to satisfy the goal with probability 1. Second, we limit how solved and not solved points are combined in the state backups. Specifically, we only combine solved points with solved points, and not solved points with not solved points. Formally, we replace W in line 3 of `Backup` with W':

$$W' = W \backslash \{w | q(s,a), q'(s',a') \in w, w \in W, \\ \text{solved}(q(s,a)) \neq \text{solved}(q'(s',a'))\}$$

If we let $J_{sol}(s)$ denote all solved points and $J_h(s)$ denote all not solved (heuristic) points for a state s, then restricting the combination of solved and not solved points will bring the complexity of computing the multi-objective Bellman equation from $O(|A(s)|(|J_{sol}(s)| + |J_h(s)|)^{|S|})$ to $O(|A(s)|(|J_{sol}(s)|^{|S|} + |J_h(s)|^{|S|}))$. Notice that because we use a single heuristic point per Pareto set and combine points in this fashion, there will only be a single partial plan expanded in line 7 of `mExpandPlan` (i.e., $\forall s |J_h(s)| = 1$).

The specific heuristic that we use is extracted from the $\mathcal{M}cLUG$ (Bryce, Kambhampati, & Smith, 2006). It estimates the cost to satisfy the goal with at least a given probability (which we set to 1.0) by computing a relaxed plan. The $\mathcal{M}cLUG$ relaxed plan estimates the conformant probabilistic plan cost, which can be seen as an additional relaxation that ignores observations. We also compare with a non-heuristic strategy where we set the cost to reach the goal with 1.0 probability to zero.

Approximating Pareto Sets: As with most multi-objective optimization problems, a Pareto set can contain an exponential number of non-dominated solutions. There exists a range of techniques for computing these Pareto sets and we explore one idea to reduce their size. It relies on the notion of ϵ-domination (Papadimitriou & Yannakakis, 2003) to prune solutions. By inflating each objective of other sub-plans by a factor of $1 + \epsilon$, it is possible to approximate the Pareto to within a relative error of ϵ in each objective by using a different definition of domination:

$q'(s,a') \prec^{\epsilon} q(s,a) \Leftrightarrow \forall_i q'_i(s,a') \leq (1+\epsilon) q_i(s,a)$

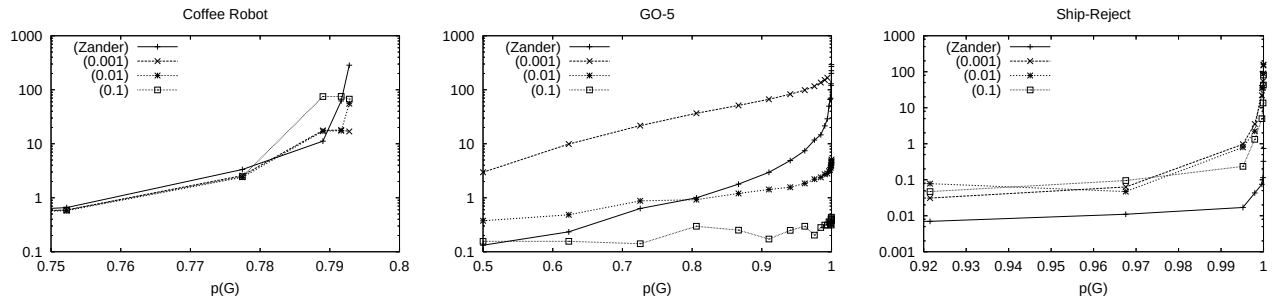

Figure 5: Run times (s) vs. τ for Coffee Robot, GO-5, and Ship-Reject.

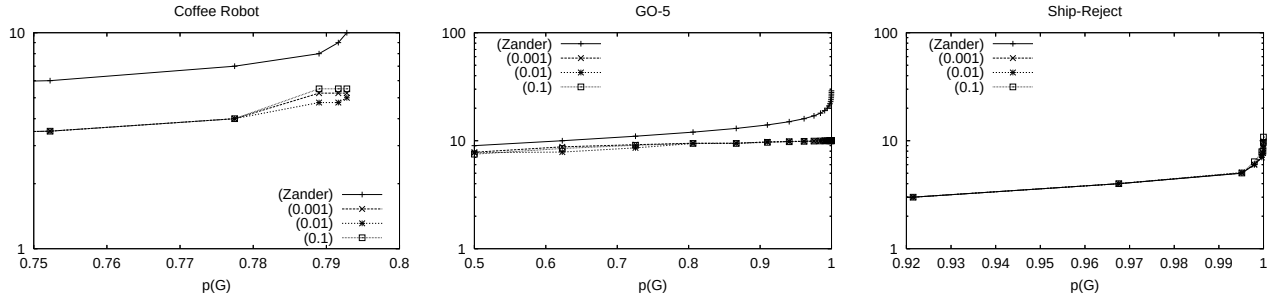

Figure 6: Expected Plan Cost vs. τ for Coffee Robot, GO-5, and Ship-Reject.

The resulting Pareto set is polynomial sized in the number of sub-plans and $\frac{1}{\epsilon}$ (c.f. Theorem 1, Papadimitriou & Yannakakis, 2003), but may still take exponential time to compute. However, because the number of non-dominated sub-plans tends to increase during bottom-up dynamic programming (i.e., a parent's Pareto set is exponential in its childrens' Pareto sets) pruning more sub-plans can have a large effect. In future work where we analyze the error in $MOLAO^*$, we intend to incorporate the error introduced by approximating the Pareto sets.

Randomized Expansions: Instead of expanding every leaf state of the current partial plan, it is possible to expand a single leaf state that is reached by simulating the partial plan. This variation of $MOLAO^*$ called $MOLAO^{*r}$ samples a single successor state s' from the transition relation in line 8 of `mExpandPlan` instead of iterating over every successor. The intuition for this variation is to concentrate plan synthesis on the most likely plan branches.

6 Empirical Results

We implemented $MOLAO^*$ within the $POND$ planner to take advantage of its reachability heuristics computed from the $\mathcal{M}cLUG$. In the following, each reference to $MOLAO^*$ is with respect to the modified version (described above). Given our description of approaches for solving conditional probabilistic planning, it seems that no previous work is directly relevant for empirical comparison. As a coarse comparison, we use the Zander planner (Majercik & Littman, 2003) in the following manner. Zander finds the maximum probability k-horizon plan with a stochastic satisfiability solver. By increasing k incrementally, it is possible to find the optimal probability of goal satisfaction τ for different expected plan lengths. Unlike our approach, the conditional plans generated by Zander require each plan branch is length k. Thus, for the same τ, $POND$ may find lower expected length plans by virtue of removing this structural assumption. In fact, $POND$ generates conditional plans as digraphs, where paths may have heterogenous lengths.

The domains that we use for comparison with Zander include Coffee Robot, GO-5, and Ship-Reject from Majercik & Littman (2003). Noticing that these domains are small (in the number of states and actions, and the plan lengths), we developed two domains: First-Responders and Grid. Both GO-5 and First-Responders plans use loops in the belief state space. The following table lists the number of Actions $|A|$, Propositions $|P|$, and observations $|O|$ in each problem.

Problem	$\|A\|$	$\|P\|$	$\|O\|$
Coffee Robot	6	8	4
GO-5	5	6	5
Ship-Reject	4	5	2
FR1	82	29	22
FR2	116	45	28
FR3	200	54	42
Grid	5	20	2

The First-Responders domain is inspired by the planning required of dispatchers in disaster scenarios. There may be multiple victims (of unknown health) that need to be treated on the scene (with a lower probability of improvement) or taken to the hospital by an ambulance (where their health will improve with high probability). There may also be fires at several locations that fire trucks can extinguish. Both fire and ambulances can report what they observe at a location. The goal is to bring each victim to health and extinguish all fires. We use three instances: FR1 has two fires, two victims, and four locations; FR2 has two fires, four victims, and four locations; and FR3 has two fires, two victims, and nine locations.

The Grid domain is an adaptation of the grid problem first described by Hyafil & Bacchus (2004) to include an action with observations. Like the original domain, a robot is moving about a 10x10 grid. The robot starts in the center of the grid and must reach a given corner. Its actions move it in the intended direction with 0.8 probability and in an adjacent direction with 0.1 probability. A sensory action allows the robot to perfectly sense the presence of a wall.

		$MOLAO^{*r}$					$MOLAO^{*h}$					$MOLAO^{*hr}$				
	τ	T(s)	E[C]	PS	DS	TS	T(s)	E[C]	PS	DS	TS	T(s)	E[C]	PS	DS	TS
FR1	0.25	314.8	9.8	101.0	3.0	19.0	41.0	12.1	165.2	14.8	20.2	14.5	12.0	42.6	1.2	13.8
	0.50	352.7	19.1	185.0	13.0	24.0	56.8	21.9	335.2	31.8	31.0	30.9	22.5	85.8	4.4	17.0
	0.75	370.2	25.6	277.0	25.0	27.0	62.1	26.8	448.2	47.2	37.4	42.4	27.5	160.2	10.0	20.8
	0.95	-	-	-	-	-	81.7	30.1	533.6	65.8	31.8	145.8	29.0	330.8	29.5	30.8
FR2	0.25	-	-	-	-	-	291.9	15.5	447.2	65.4	39.0	160.3	18.0	86.6	2.6	17.2
	0.50	-	-	-	-	-	318.6	26.3	469.4	53.4	31.4	130.9	24.7	115.4	5.6	19.6
	0.75	-	-	-	-	-	370.1	32.8	585.4	77.8	37.0	231.9	31.1	218.6	17.4	25.4
	0.95	-	-	-	-	-	457.4	39.3	816.8	98.8	41.2	292.2	37.5	373.0	35.0	30.0
FR3	0.25	-	-	-	-	-	420.9	12.8	392.0	34.2	33.0	252.8	15.0	69.0	1.8	15.2
	0.50	-	-	-	-	-	468.2	23.5	553.3	50.0	39.3	265.0	27.0	141.6	4.4	20.2
	0.75	-	-	-	-	-	623.7	40.67	741.0	51.0	45.0	459.2	35.3	248.0	13.0	26.4
	0.95	-	-	-	-	-	624.6	39.2	865.0	70.0	49.0	-	-	-	-	-
Grid	0.25	-	-	-	-	-	32.3	10.8	21.2	0.0	6.8	21.7	13.2	19.8	0.0	6.8
	0.50	-	-	-	-	-	356.7	19.4	153.2	0.2	36.8	55.5	19.0	54.8	0.2	14.8
	0.75	-	-	-	-	-	954.5	23.6	537.6	0.8	104.8	161.3	21.5	150.6	0.0	29.8

Table 1: Results using $MOLAO^{*r}$, $MOLAO^{*h}$, and $MOLAO^{*hr}$.

Zander Domains: The plots in Figures 5 and 6 show results for the Coffee-Robot, GO-5, and Ship-Reject domains. We compare Zander with $MOLAO^*$ without a heuristic or randomized expansions by varying the value of ϵ for approximate Pareto sets. Using the $\mathcal{M}cLUG$ heuristic in these problems showed little improvement. The plot legends indicate the results for $POND$ by the value of ϵ. The total planning time in seconds is shown in Figure 5 and the expected plan length is shown in Figure 6 for several values of τ. Despite $POND$ not guaranteeing optimal plans, it finds plans that are less than or equal to the expected length of plans found by Zander (per the discussion above). In terms of planning time, in most cases $POND$ performs better than Zander as ϵ increases (decreasing the size of J-sets). Zander performs better in the Ship-Reject; the $POND$ and Zander plan costs are close, indicating that Zander is better when are balanced trees. $POND$ is able to capitalize on the plans that are unbalanced trees in other domains.

New Domains: Results for the First-Responders and Grid domain are shown in Table 1. All the results are the average of 5 runs. We compare three versions of the modified $MOLAO^*$ where the superscript "r" corresponds to using randomized expansions, and the superscript "h" corresponds to using the relaxed plan heuristic (instead of zero) for heuristic points. In each version we use $\epsilon = 0.1$ and 10 samples within each $\mathcal{M}cLUG$ (Bryce, Kambhampati, & Smith, 2006). We do not show results for the version of modified $MOLAO^*$, used in the Zander domains, because it was not able to solve any of the instances for any value of τ. Due to complications with encoding the domains

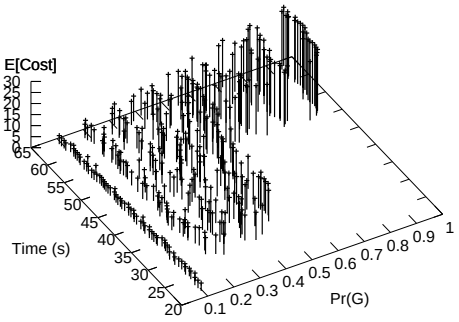

Figure 7: E[C] vs. τ over time on FR1, using $MOLAO^{*h}$.

in Zander's input format, we do not report Zander results. We expect Zander to perform comparable to $MOLAO^*$ without heuristics. The table reports total time in seconds "T(s)", plan cost "E[C]", number of unique belief states in the plan where an action is executed "PS", number of belief states with more than one parent in the plan "DS", and the number of terminal belief states "TS" .

We see that the probabilistic conformant planning graph heuristic does indeed improve scalability, as we hoped. $MOLAO^{*r}$ is only able to solve the smallest First-Responder instance to a 0.75 probability of goal satisfaction. However, using the heuristic in $MOLAO^{*h}$ and $MOLAO^{*hr}$ allows $POND$ to solve every instance to a high level of probability. While the expected cost is slightly better without using the heuristic, we see the benefit of using an inadmissible, but informative heuristic. In the instances requiring the heuristic, the number of plan belief states (PS) is quite large, testifying to the scale of the plans. The number of plan belief states with more than one parent (DS) shows the benefit of using $MOLAO^*$ to find more compact digraph plans. The number of terminal belief states (TS) attests to the number of branches in the plans (modulo the number of collapsed branches, captured by DS).

While we are interested in solving the conditional probabilistic planning problem, we can also find Pareto sets of plans. Figure 7 shows the Pareto set for b_I over time (after each iteration in `ExpandPlan`) in the FR1 domain while using $MOLAO^{*h}$. Each line represents a solution satisfying the goal with the indicated probability and cost. The set of feasible plans steadily increase the probability of goal satisfaction over time, suggesting how $MOLAO^*$ can be used in an anytime fashion to find a Pareto set of plans.

7 Limited Contingency Planning

Limited contingency planning involves bounding the number of plan branches k to make plans more human-understandable and compact (Meuleau & Smith, 2003). Meuleau & Smith formulate k-contingency problems as a POMDP where they embed the number of contingencies as a state feature and make k copies of the state space. Executing a sensory action transitions the state to a logically equivalent state where there are fewer available contingencies. A sensory action cannot be executed in states where it introduces more than the available number of contingencies. The grid problem, which has 100 reachable states (2^{20} total states), could be encoded as a POMDP with $100(k+1)$ states. However, this problem proved difficult for a POMDP solver when $k = 1$ (Hyafil & Bacchus, 2004), making it doubtful that it could scale with k.

In $MOLAO^*$, we can trade this state space explosion for a third objective $q_2(s, a)$. The objective is computed as $q_2(s, a) = \sum_{s' \in S: T(s,a,s') > 0} q'_2(s', a')$ and $q_2(s, \perp) = 1$. We can identify plans with k or less branches when $q_2(s_I, a) \leq k$. Furthermore, we can prune a point $q(s, a)$ at any time if $q_2(s, a) > k$ because its sub-plan already exceeds the number of contingencies. We initialize all heuristic points as $q_2(s, *) = 1$ because all plans have at least one contingency. While the number of contingencies can be seen as an objective, we assume that all numbers of contingencies less than k are equally preferred and use the objective as a hard constraint for plan feasibility rather than optimization.

Table 2 presents results from the Grid domain, where we vary the maximum number of branches k allowable in a feasible plan. As we can see, $MOLAO^{*h}$ scales reasonably well with k. Using k as a pruning condition greatly improves time to find a plan compared to the unbounded case.

	τ = 0.25			τ = 0.5			τ = 0.75		
k	T(s)	E[C]	TS	T(s)	E[C]	TS	T(s)	E[C]	TS
2^0	10.4	10.7	1.0	11.9	13.7	1.0	14.1	18.0	1.0
2^1	16.2	10.9	1.5	21.8	15.0	1.2	25.8	20.5	1.0
2^2	20.9	11.1	1.5	26.5	13.6	2.0	46.5	19.4	1.7
2^3	19.1	11.1	3.0	35.3	14.7	3.7	73.0	20.6	3.0
2^6	43.8	16.3	2.8	77.8	23.4	10.6	110.5	24.9	9.8
2^9	56.4	17.0	3.8	47.7	23.2	5.4	245.5	29.1	43.0
2^{12}	52.7	18.7	4.4	69.2	27.1	9.8	146.9	30.8	23.6

Table 2: Limited contingency $MOLAO^{*h}$ on Grid.

8 Related Work

Our formulation of conditional probabilistic planning is based on existing work techniques for vector-valued MDPs (Henig, 1983). We define cost, probability of goal satisfaction, and number of branches as values optimized in the MDP. Where previous work concentrates on value iteration, we extend the more appropriate LAO^* algorithm to deal with multiple objectives.

As mentioned earlier, there are several techniques for solving conditional probabilistic planning. Earliest work (Draper, Hanks, & Weld, 1994; Onder, 1999) used partial order planning techniques to find feasible plans (ignoring plan cost). In parallel, work on single objective MDPs and POMDPs (surveyed by Boutilier, Dean, & Hanks (1999)) were developed to find cost optimal plans (using penalties for not satisfying the goal). More recently, Majercik & Littman (2003) formulated the problem as stochastic satisfiability to find the optimal probability of goal satisfaction for finite horizon plans. None of the previous works use reachability heuristics to guide plan synthesis and hence have concentrated on relatively small problems.

9 Conclusion

We have shown that embracing the multi-objective nature of probabilistic planning allows us to overcome restrictive assumptions made in single objective formulations. It also gives us a richer notion of conditional plans: providing a Pareto set of plans (and sub-plans) to the plan executor introduces additional "choice" contingencies to the set of nature selected "chance" contingencies. These benefits incur a greater combinatorial overhead, but are offset by ϵ-domination techniques for Pareto set approximation. Further, because our formulation uses cost as an optimization criterion, we can make use of effective planning graph reachability heuristics that estimate plan cost. We have also shown that additional objectives, such as the number of plan branches in limited contingency planning, are easily integrated within our framework.

Future work will include a more formal analysis of the $MOLAO^*$ search algorithm and alternative representations and algorithms for computing Pareto sets.

References

Barto, A. G.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *AIJ* 72:81–138.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006. Sequential monte carlo in probabilistic planning reachability heuristics. In *Proceedings of ICAPS'06*.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In *Proceedings of AIPS-94*.

Hansen, E., and Zilberstein, S. 2001. LAO: A heuristic-search algorithm that finds solutions with loops. *AIJ* 129(1–2):35–62.

Henig, M. 1983. Vector-valued dynamic programming. *Siam J. Cont.* 21:490–499.

Hyafil, N., and Bacchus, F. 2004. Utilizing structured representations and CSPs in conformant probabilistic planning. In *Proceedings of ECAI'04*, 1033–1034.

Kushmerick, N.; Hanks, S.; and Weld, D. 1994. An algorithm for probabilistic least-commitment planning. In *Proceedings of AAAI'94*, 1073–1078.

Little, I.; Aberdeen, D.; and Theibaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proc. of AAAI'05*, 1181–1186.

Littman, M. 1997. Probabilistic propositional planning: Representations and complexity. In *Proceedings of AAAI'97*.

Majercik, S., and Littman, M. 2003. Contingent planning under uncertainty via stochastic satisfiability. *AIJ* 147(1-2):119–162.

Meuleau, N., and Smith, D. 2003. Optimal limited contingency planning. In *Proceedings of UAI-03*.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Morgan Kaufmann.

Onder, N. 1999. *Contingency Selection in Plan Generation*. Ph.D. Dissertation, University of Pittsburgh.

Papadimitriou, C. H., and Yannakakis, M. 2003. The complexity of tradeoffs, and optimal access of web sources. In *Proc. of FOCS-03*.

Wu, J., and Azram, S. 2001. Metrics for quality assessment of a multiobjective design optimization solution set. *Journal of Mechanical Design* 123:18–25.