A Hybridized Planner for Stochastic Domains

Mausam
Dept. of Computer Sc. and Engg.
University of Washington
Seattle, WA-98195
mausam@cs.washington.edu

Piergiorgio Bertoli
ITC-IRST
via Sommarive 18
38050 Povo, Trento, Italy
bertoli@itc.it

Daniel S. Weld
Dept. of Computer Sc. and Engg.
University of Washington
Seattle, WA-98195
weld@cs.washington.edu

Abstract

Markov Decision Processes are a powerful framework for planning under uncertainty, but current algorithms have difficulties scaling to large problems. We present a novel probabilistic planner based on the notion of *hybridizing* two algorithms. In particular, we hybridize GPT, an exact MDP solver, with MBP, a planner that plans using a qualitative (non-deterministic) model of uncertainty. Whereas exact MDP solvers produce optimal solutions, qualitative planners sacrifice optimality to achieve speed and high scalability. Our hybridized planner, HYBPLAN, is able to obtain the best of both techniques — speed, quality and scalability. Moreover, HYBPLAN has excellent *anytime* properties and makes effective use of available time and memory.

1 Introduction

Many real-world domains involve uncertain actions whose execution may stochastically lead to different outcomes. Such problems are frequently modeled as an indefinite horizon Markov Decision Process (MDP) also known as stochastic shortest path problem [Bertsekas, 1995]. While MDPs are a very general framework, popular optimal algorithms (e.g., LAO* [Hansen and Zilberstein, 2001], Labeled RTDP [Bonet and Geffner, 2003]) do not scale to large problems.

Many researchers have argued for planning with a qualitative, non-deterministic model of uncertainty (in contrast to numeric probabilities). Such contingent planners (*e.g.*, MBP [Bertoli *et al.*, 2001]) cannot make use of quantitative likelihood (and cost) information. Because they solve a much simpler problem, these planners are able to scale to much larger problems than probabilistic planners. By stripping an MDP of probabilities and costs, one could use a qualitative contingent planner to quickly generate a policy, but the quality of the resulting solution will likely be poor. Thus it is natural to ask "can we develop an algorithm with the benefits of both frameworks?".

Using two (or more) algorithms to obtain the benefits of both is a generic idea that forms the basis of many proposed algorithms, *e.g.*, bound and bound [Martello and Toth, 1990] and algorithm portfolios [Gomes and Selman, 2001]. Various schemes hybridize multiple algorithms differently and are aimed at different objectives. For example, algorithm portfolios run multiple algorithms in parallel and thus reduce the total time to obtain a solution; bound and bound uses a solution from one algorithm as a bound for the other algorithm. In this paper we employ a tighter notion of hybridization, where we explicitly incorporate solutions to sub-problems from one algorithm into the partial solution of the other.

We present a novel algorithm, HYBPLAN, which hybridizes two planners: GPT and MBP. GPT (General Planning Tool) [Bonet and Geffner, 2005] is an exact MDP solver using labeled real time dynamic programming (RTDP). MBP (Model Based Planner) [Bertoli *et al.*, 2001], is a non-deterministic planner exploiting binary decision diagrams (BDDs). Our hybridized planner enjoys benefits of both algorithms, *i.e.*, *scalability* and *quality*. To the best of our knowledge, this is the first attempt to bridge the efficiency-expressiveness gap between the planners dealing with qualitative *vs.* probabilistic representations of uncertainty.

HYBPLAN has excellent *anytime* properties — it produces a legal solution very fast and then successively improves on this solution[1]. Moreover, HYBPLAN can effectively handle interleaved planning and execution in an online setting, makes use of the available time and memory efficiently, is able to converge within a desired optimality bound, and reduces to optimal planning given an indefinite time and memory. Our experiments demonstrate that HYBPLAN is competitive with the state-of-the-art planners, solving problems in the International Planning Competition that most other planners could not solve.

2 Background

Following [Bonet and Geffner, 2003], we define an indefinite horizon Markov decision process as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}r, \mathcal{C}, \mathcal{G}, s_0 \rangle$:

- $\mathcal{S}$ is a finite set of discrete states. We use factored MDPs, *i.e.*, $\mathcal{S}$ is compactly represented in terms of a set of state variables.
- $\mathcal{A}$ is a finite set of actions. An applicability function, $Ap : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, denotes the set of actions that can be applied in a given state ($\mathcal{P}$ represents the power set).

[1] While RTDP is popular as an anytime algorithm itself, for problems with absorbing goals it may not return any legal policy for as much as 10 min. or more.

- $\mathcal{P}r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function. $\mathcal{P}r(s'|s, a)$ denotes the probability of arriving at state s' after executing action a in state s.
- $\mathcal{C} : \mathcal{A} \rightarrow \Re^+$ is the cost model.
- $\mathcal{G} \subseteq \mathcal{S}$ is a set of absorbing goal states, *i.e.*, the process ends once one of these states is reached.
- s_0 is a start state.

We assume full observability, and we seek to find an optimal stationary policy, *i.e.*, a function π: $\mathcal{S} \rightarrow \mathcal{A}$, which minimizes the expected cost (over an indefinite horizon) incurred to reach a goal state. A policy π and its execution starting from the start state induces an execution structure *Exec*[π]: a directed graph whose nodes are states from $\mathcal{S}$ and transitions are labeled with the actions performed due to π. We denote this graph to be *free of absorbing cycles* if for every non-goal node there always exists a path to reach a goal (*i.e.*, the probability to reach a goal is always greater than zero). A policy free of absorbing cycles is also known as a *proper* policy.

Note that any *cost function*, J: $\mathcal{S} \rightarrow \Re$, mapping states to the expected cost of reaching a goal state defines a *greedy policy* as follows:

$$\pi_J(s) = \operatorname*{argmin}_{a \in Ap(s)} \left\{ \mathcal{C}(a) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s, a) J(s') \right\} \quad (1)$$

The *optimal* policy derives from a value function, J^*, which satisfies the following pair of *Bellman equations*.

$$J^*(s) = 0, \text{ if } s \in \mathcal{G} \text{ else}$$
$$J^*(s) = \min_{a \in Ap(s)} \left\{ \mathcal{C}(a) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s, a) J^*(s') \right\} \quad (2)$$

2.1 Labeled RTDP

Value iteration is a canonical algorithm to solve an MDP. In this dynamic programming approach the optimal value function (the solution to equations 2) is calculated as the limit of a series of approximations, each considering increasingly long action sequences. If $J_n(s)$ is the value of state s in iteration n, then the value of state s in the next iteration is calculated with a process called a *Bellman backup* as follows:

$$J_{n+1}(s) = \min_{a \in Ap(s)} \left\{ \mathcal{C}(a) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s, a) J_n(s') \right\}$$

Value iteration terminates when $\forall s \in \mathcal{S}$, $|J_n(s) - J_{n-1}(s)| \leq \epsilon$, and this termination is guaranteed for $\epsilon > 0$. Furthermore, the sequence of $\{J_i\}$ is guaranteed to converge to the optimal value function, J^*, regardless of the initial values. Unfortunately, value iteration tends to be quite slow, since it explicitly updates every state, and $|\mathcal{S}|$ is exponential in the number of domain features. One optimization restricts search to the part of state space reachable from the initial state s_0. Two algorithms exploiting this *reachability analysis* are LAO* [Hansen and Zilberstein, 2001] and our focus: RTDP [Barto *et al.*, 1995].

RTDP, conceptually, is a lazy version of value iteration in which the states get updated in proportion to the frequency with which they are visited by the repeated executions of the greedy policy. Specifically, RTDP simulates the greedy policy along a single trace execution, and updates the values of the states it visits using Bellman backups. An RTDP *trial* is a path starting from s_0 and ending when a goal is reached or the number of updates exceeds a threshold. RTDP repeats these trials until convergence. Note that common states are updated frequently, while RTDP wastes no time on states that are unreachable, given the current policy. The complete convergence (at every state) is slow because less likely (but potentially important) states get updated infrequently. Furthermore, RTDP is not guaranteed to terminate.

Labeled RTDP fixes these problems with a clever labeling scheme that focuses attention on states where the value function has not yet converged [Bonet and Geffner, 2003]. Specifically, states are gradually labeled *solved* signifying that the value function has converged for them. The algorithm back-propagates this information starting from the goal states and the algorithm terminates when the start state gets labeled. Labeled RTDP is guaranteed to terminate. It is guaranteed to converge to the optimal value function (for states reachable using the optimal policy), if the initial value function is admissible. This algorithm is implemented in GPT (General Purpose Tool) [Bonet and Geffner, 2005].

(Labeled) RTDP is popular for quickly producing a relatively good policy for *discounted* reward-maximization problems, since the range of infinite horizon total rewards is finite. However, problems in the planning competition are undiscounted cost minimization problems with absorbing goals. Intermediate RTDP policies on these problems often contain absorbing cycles. This implies that the expected cost to reach the goal is infinite! Clearly, RTDP is *not* an anytime algorithm for our problems because ensuring that all trajectories reach a goal takes a long time.

2.2 The Model-Based Planner

An MDP *without* cost and probability information translates into a planning problem with qualitative uncertainty. A strong-cyclic solution to this problem — one that admits loops but is free of absorbing cycles — can be used as a legal solution to the original MDP, though it may be highly sub-optimal. However, because we are solving a much easier problem, the algorithms for solving this relaxed problem are highly scalable. One such algorithm is implemented within MBP.

The Model-Based Planner, MBP [Bertoli *et al.*, 2001], relies on effective BDD-based representation techniques to implement a set of sound and complete plan search and verification algorithms. All the algorithms within MBP are designed to deal with qualitatively non-deterministic planning domains, defined as Moore machines using MBP's input language SMV. The planner is very general and is capable of accommodating domains with various state observability (*e.g.*, fully observable, conformant) and different kinds of planning goals (*e.g.*, reachability goals, temporal goals).

MBP has two variants of strong-cyclic algorithm denoted by "global" and "local". Both share the representation of the

policy π as a binary decision diagram, and the fact that it is constructed by backward chaining from the goal. The general idea is to iteratively regress from a set of solution states (the current policy π), first admitting any kind of loop introduced by the backward step, and then removing those "bad" loops for which no chance of goal achievement exists. On top of this, the "local" variant of the algorithm prioritizes solutions with no loops, in order to retrieve strong solutions whenever possible. The search ends either when π covers the initial state, or when a fixed point is reached. Further details can be found in [Cimatti *et al.*, 2003].

3 HYBPLAN: A Hybridized Planner

Our novel planner, HYBPLAN, hybridizes GPT and MBP. On the one hand, GPT produces cost-optimal solutions to the original MDP; on the other, MBP ignores probability and cost information, but produces *a* solution extremely quickly. HYBPLAN combines the two to produce high-quality solutions in intermediate running times.

At a high level, HYBPLAN invokes GPT with a maximum amount of time, say *hybtime*. GPT preempts itself after running for this much time and passes the control back to HYBPLAN. At this point GPT has performed several RTDP trials and might have labeled some states solved. However, the whole cost function J_n has not converged and the start state is not yet solved. Despite this, the current greedy partial[2] policy (as given by Equation 1) contains much useful information. HYBPLAN combines this partial policy (π_{GPT}) with the policy from MBP (π_{MBP}) to construct a *hybridized* policy (π_{HYB}) that is defined for all states reachable following π_{HYB}, and is guaranteed to lead to the goal. We may then evaluate π_{HYB} by computing $J_{\text{HYB}}(s_0)$ denoting the expected cost to reach a goal following this policy. In case we are dissatisfied with π_{HYB}, we may run some more RTDP trials and repeat the process. We describe the pseudo-code for the planner in Algorithm 1.

The construction of the hybridized policy starts from the start state and uses the popular combination of *open* and *closed* lists denoting states for which an action needs to be assigned and have been assigned, respectively. Additionally we maintain a *deadends* list that memoizes all states starting from which we cannot reach a goal (dead-end).

Deciding between GPT and MBP (lines 9 to 15): For every state s, HYBPLAN decides whether to assign the action using π_{GPT} or π_{MBP}. If s is already labeled *solved* then we are certain that GPT has computed the optimal policy starting from s (lines 8-9). Otherwise, we need to assess our confidence in $\pi_{\text{GPT}}(s)$. We estimate our confidence on GPT's greedy policy by keeping a count on the number of times s has been updated inside labeled RTDP. Intuitively, smaller number of visits to a state s corresponds to our low confidence on the quality of $\pi_{\text{GPT}}(s)$ and we may prefer to use $\pi_{\text{MBP}}(s)$ instead. A user-defined *threshold* decides on how quickly we start trusting GPT.

MBP returning with failure (Function 2): Sometimes MBP may return with a failure implying that no solution exists from the current state. Clearly, the choice of the action (in the previous step) is faulty because that step led to this dead-end. The procedure ASSIGNMBPSOLUTION recursively looks at the policy assignments at previous levels and debugs π_{HYB} by assigning solutions from π_{MBP}. Additionally we memoize the dead-end states to reduce future computation.

[2]It is partial because some states reachable by the greedy policy might not even be explored yet.

Algorithm 1 HYBPLAN(*hybtime, threshold*)

```
 1: deadends ← ∅
 2: repeat
 3:    run GPT for hybtime
 4:    open ← ∅; closed ← ∅
 5:    open.insert(s_0)
 6:    while open is non-empty do
 7:       remove s from open
 8:       closed.insert(s)
 9:       if s is labeled solved inside GPT then
10:          π_HYB(s) ← π_GPT(s)
11:       else
12:          if visit(s) > threshold then
13:             π_HYB(s) ← π_GPT(s)
14:          else
15:             AssignMbpSolution(s)
16:       for all s' s.t. Pr(s'|s, π_HYB(s)) > 0, s' ∉ closed do
17:          if s' ∈ deadends then
18:             AssignMbpSolution(s)
19:          else
20:             open.insert(s')
21:    remove absorbing cycles from Exec[π_HYB]
22:    evaluate π_HYB by computing J_HYB(s_0)
23:    if π_HYB is the best policy found so far, cache it
24: until all resources exhaust or desired error bound is achieved
```

Function 2 ASSIGNMBPSOLUTION(s)

```
if MBP(s) succeeds then
   π_HYB(s) ← π_MBP(s)
   for all s' s.t. Pr(s'|s, π_HYB(s)) > 0, s' ∉ closed do
      open.insert(s')
else
   if s = s_0 then
      problem is unsolvable; exit()
   else
      closed.remove(s)
      deadends.insert(s)
      for all s' s.t. π_HYB(s') leads directly to s do
         AssignMbpSolution(s')
```

Cleaning, Evaluating and Caching π_{HYB} (lines 21-23): We can formulate the evaluation of π_{HYB} as the following system of linear equations:

$$J_{\text{HYB}}(s) = 0, \text{ if } s \in \mathcal{G} \text{ else}$$
$$J_{\text{HYB}}(s) = \mathcal{C}(\pi(s)) + \sum_{s' \in \mathcal{S}} \mathcal{P}r(s'|s, \pi(s)) J_{\text{HYB}}(s') \quad (3)$$

These equations are tricky to solve, because it is still possible that there is an absorbing cycle in *Exec*[π_{HYB}]. If the rank of the coefficient matrix is less than the number of equations then there is an absorbing cycle. We can convert the matrix

into row-echelon form by using row transformations and in parallel perform the same transformations on the identity matrix. When we find a row with all zeros the non-zero entries of the same row in the transformed identity matrix reveals the states in the original system that form absorbing cycle(s). We pick one of these states, assign the MBP action for it and repeat this computation. Note that this system of equations is on a very small fraction of the overall state space (which are in $Exec[\pi_{\text{HYB}}]$), hence this step is not expensive. As we find intermediate hybridized policies, we cache the best (*i.e.*, the one with minimum $J_{\text{HYB}}(s_0)$) policy found so far (line 23).

Termination (line 24): We may terminate differently in different situations: given a fixed amount of time, we may stop GPT when the available time is about to expire and follow it with a hybridized policy computation and terminate. Given a fixed amount of memory, we may do the same with memory. If we need to terminate within a desired fraction of the optimal, we may repeat hybridized policy computation at regular intervals and when we find a policy whose error bound is within the desired limits, we terminate.

Error Bound: We develop a simple procedure to bound the error in π_{HYB}. Since labeled RTDP is always started with an admissible heuristic, GPT's current cost function J_n remains a lower bound of the optimal ($J_n \leq J^*$). However the hybridized policy is clearly worse than the optimal and hence $J_{\text{HYB}} \geq J^*$. Thus, $\frac{J_{\text{HYB}}(s_0)-J_n(s_0)}{J_n(s_0)}$ bounds the error of π_{HYB}.

Properties of HYBPLAN: HYBPLAN uses the current greedy policy from GPT, combines it with solutions from MBP for states that are not fully explored in GPT and ensures that the final hybridized policy is *proper*, *i.e.*, free of absorbing cycles. Thus HYBPLAN has excellent anytime properties, *i.e.*, once $\pi_{\text{MBP}}(s_0)$ has returned with success, HYBPLAN is capable of improving the quality of the solution as the time available for the algorithm increases. If infinite time and resources are available for the algorithm, then the algorithm reduces to GPT, whereas if the available resources are extremely limited, then it reduces to MBP. In all other cases, the hybridized planner demonstrates intermediate behavior.

3.1 Two Views of the Hybridized Planner

Our hybridized planner may be understood in two ways. The first view is MBP-centric. If we run HYBPLAN without any GPT computation then only π_{MBP} will be outputted. This solution will be a legal but possibly low quality. HYBPLAN successively improves the quality of this basic solution from MBP by plugging in additional information from GPT.

An alternative view is GPT-centric. We draw from the intuition that, in GPT, the partial greedy policy (π_{GPT}) improves gradually and eventually gets defined for all relevant states accurately. But before convergence, the current greedy policy may not even be defined for many states and may be inaccurate for others, which have not been explored enough. HYBPLAN uses this partial policy as much as reasonable and completes it by adding in solutions from MBP; thus making the final policy consistent and useful. In essence, both views are useful: each algorithm patches the other's weakness.

3.2 Implementation of HYBPLAN

We now address two different efficiency issues in implementing HYBPLAN. First, instead of pre-computing an MBP policy for the whole state space, we do this computation on demand. We modify MBP so that it can efficiently solve the sub-problems without repeating any computation. We modify MBP's "local" strong cyclic planning algorithm in the following ways — 1) we cache the policy table (π_{MBP}) produced by previous planning episodes, 2) at each planning episode, we analyze the cached result π_{MBP}, and if the input state is solved already, search is skipped, 3) we perform search by taking π_{MBP} as a starting point (rather than the goal).

Second, in our implementation we do not evaluate π_{HYB} by solving the system of linear equations. Instead, we approximate this by averaging repeated simulations of the policy from the start state. If any simulation exceeds a maximum trajectory length we guess that the hybrid policy has an absorbing cycle and we try to break the cycle by recursively assigning the action from π_{MBP} for a state in the cycle. This modification speeds up the overall algorithm. Although in theory this takes away the guarantee of reaching the goal with probability 1 since there could be some low probability trajectory not explored by the simulation that may contain an absorbing cycle, in practice this modification is sufficient as even the planning competition relies on policy simulation for evaluating planners. In our experiments, our hybridized policies reach the goal with probability 1.

We finally remark that while GPT takes as input a planning problem in probabilistic PDDL format, MBP's input is a domain in SMV format. Our translation from PDDL to SMV is systematic but only semi-automated; we are implementing a fully automated procedure.

4 Experiments

We evaluate HYBPLAN on the speed of planning, quality of solutions returned, anytime behavior, and scalability to large problems. We also perform a sensitivity experiment testing the algorithm towards sensitivity to the parameters.

Methodology

We compare HYBPLAN with GPT and MBP. In the graphs we plot the expected cost of the cached policy for both HYBPLAN and GPT as a function of time. The first value in the HYBPLAN curve is that of MBP (since initially for each state s, *visit*(s) = 0, and thus $\pi_{\text{HYB}} = \pi_{\text{MBP}}$). We also plot the current $J_n(s_0)$ value from labeled RTDP. As this admissible value increases, the error bound of the solution reduces.

We run the experiments on three large probabilistic PDDL domains. The first two domains are probabilistic variants of the Rovers and MachineShop domains from the 2002 AIPS Planning Competition. The third is the Elevators domain from the 2006 ICAPS Planning Competition. The largest problem we attempted was in the Elevators domain and had 606 state variables.

For our experiments we terminate when either labeled RTDP terminates or when the memory goes out of bound. For most experiments, we initialize HYBPLAN with *hybtime* = 25 sec and *threshold* = 50. We also perform experiments to analyze the sensitivity to these parameters.

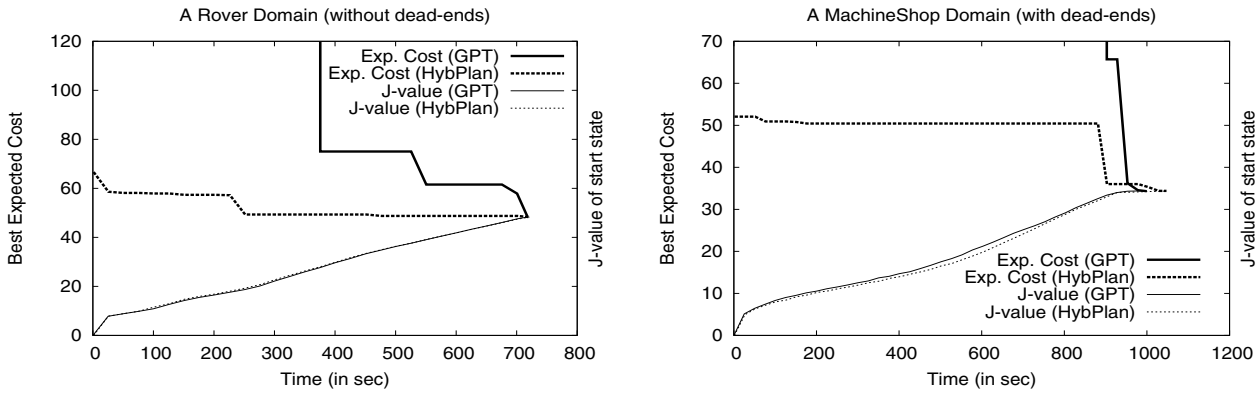

Figure 1: Anytime properties of HYBPLAN: On one Y-axis we show the expected cost of the cached policy and on the other the $J_n(s_0)$ values. J_n converges when the two curves meet. We find that HYBPLAN's policy is superior to GPT's greedy policy. The first time when GPT's policy has non-infinite expected cost occurs much later in the algorithm.

Anytime Property

We first evaluate the anytime properties of HYBPLAN for moderate sized planning problems. We show results on two problems, one from the Rovers domain and the other from the MachineShop domain (Figures 1(a), 1(b)). These problems had 27 state variables and about 40 actions each. We observe that π_{HYB} has a consistently better expected cost than did π_{GPT}, and that the difference between the two algorithms is substantial. For example, in Figure 1(b) the first time, when π_{GPT} has a non-infinite expected cost (*i.e.*, all simulated paths reach the goal), is after 950 seconds; whereas HYBPLAN always constructs a valid policy.

Figure 1(a) is for a domain without any dead-ends whereas in the domain of Figure 1(b) there are some 'bad' actions from which the agent can never recover. While HYBPLAN obtains greater benefits for domains with dead-ends, for all the domains the anytime nature of HYBPLAN is superior to the GPT. Also notice the $J_n(s_0)$ values in Figure 1. HYBPLAN sometimes takes marginally longer to converge because of overheads of hybrid policy construction. Clearly this overhead is insignificant.

Recall that the first expected cost of π_{HYB} is the expected cost of following the MBP policy. Clearly an MBP policy is not of very high quality and is substantially improved as time progresses. Also, the initial policy is computed very quickly.

Scaling to Large Problems

If we desire to run the algorithm until convergence then HYBPLAN is no better than GPT. For large problems, however, running until convergence is not a practical option due to limited resources. For example, in Figure 2 we show experiments on a larger problem from the rovers domain where the memory requirements exceed our machine's 2 GB. (typically, the memory fills up after the algorithm explores about 600,000 states) In such cases hybridization provides even more benefits (Table 3). For many of the large domains GPT is in fact unable to output a single policy with finite expected cost. While one might use MBP directly for such problems, by hybridizing the two algorithms we are able to get consistently higher quality solutions.

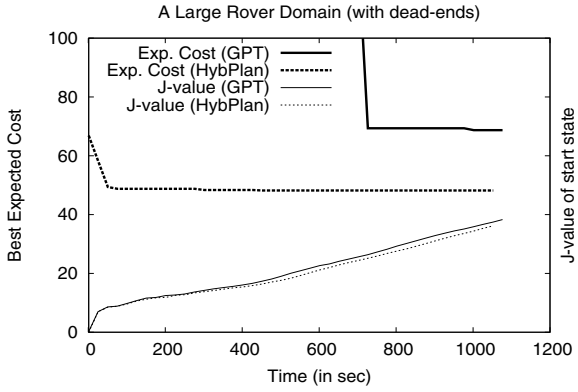

Figure 2: Plot of expected cost on a problem too large for GPT to converge.

Notice the Elevator problems in Table 3. There are 606 variables in this domain. These problems were the largest test problems in the Elevators domain for the Planning Competition 2006 and few planners could solve it. Thus HYBPLAN's performance is very encouraging.

Sensitivity to Parameters

HYBPLAN is controlled by two parameters: *hybtime* and *threshold*. We evaluate how sensitive HYBPLAN is to these parameters. We find that increasing *hybtime* reduces the total algorithm time but the difference is marginal, implying that the overhead of hybrid policy construction is not significant. However, smaller values result in repeated policy construction and this helps in finding a good quality solution early. Thus small values of *hybtime* are overall more effective.

Varying *threshold* does not affect the overall algorithm time. But it does marginally affect the first time a good solution is observed. Increasing *threshold* implies that an MBP policy is used until a state is sufficiently explored in GPT. For Rovers domain this translates to some extra time before a good policy is observed. For MachineShop we observe the

Problems	Time before memory exhausts	Expected Cost		
		GPT	MBP	HYBPLAN
Rover5	$\sim$ 1100 sec	55.36	67.04	48.16
Rover2	$\sim$ 800 sec	∞	65.22	49.91
Mach9	$\sim$ 1500 sec	143.95	66.50	48.49
Mach6	$\sim$ 300 sec	∞	71.56	71.56
Elev14	$\sim$ 10000 sec	∞	46.49	44.48
Elev15	$\sim$ 10000 sec	∞	233.07	87.46

Figure 3: Scalability of HYBPLAN: Best quality solutions found (before memory exhausts) by GPT, MBP and HYBPLAN for large problems. HYBPLAN outperforms the others by substantial margins.

opposite behavior suggesting that we are better off using MBP policies for less explored regions of the space. While overall the algorithm is only marginally sensitive to this parameter, the differences in two domains lead us to believe that the optimal values are domain-dependent and in general, intermediate values of *threshold* are preferable.

Finally, we also compare with symbolic LAO* [Feng and Hansen, 2002] to determine whether our speedup is due primarily to the use of a qualitative uncertainty representation or instead to exploitation of symbolic techniques. We observe that for our large problems symbolic LAO* does not converge even after many hours, since backing up the whole ADD takes a huge amount of time. Thus, we conclude that the speedup is due to hybridization with a simpler, qualitative model of uncertainty.

5 Discussion and Related Work

Hybridizing planners were introduced recently in the planning community [Mausam and Weld, 2005; McMahan *et al.*, 2005]. Mausam and Weld used hybridization in the context of concurrent probabilistic temporal planning by hybridizing interwoven epoch and aligned epoch planning. McMahan *et al.* used a sub-optimal policy to achieve policy guarantees in an MDP framework (an extension of bound and bound to MDPs). However, they did not provide any method to obtain this sub-optimal policy. We are the first to present a principled, automated hybridized planner for MDPs.

The strength of our work is in the coupling of a probabilistic and qualitative contingent planner. While there have been several attempts to use *classical* planners to obtain probabilistic solutions (*e.g.*, generate, test and debug [Younes and Simmons, 2004], FF-Replan), they have severe limitations because conversion to classical languages results in decoupling various outcomes of the same action. Thus the classical planners have no way to reject an action that has two outcomes – one good and one bad. However, this kind of information is preserved in domains with qualitative uncertainty and combining them with probabilistic planners creates a time-quality balanced algorithm.

Although the computation of a proper policy is a strength of HYBPLAN, it is also a weakness because there exist problems (which we term "improper"), which may not contain even a single proper policy. For these improper problems the algorithm (in its present form) will deem the problem unsolvable, because we have specifically chosen GPT and the strong-cyclic planning algorithm of MBP. Instead, we could hybridize other algorithms (*e.g.*, PARAGRAPH [Little and Thiebaux, 2006] or GPT supplemented with a high but non-infinite cost of reaching a dead-end) with MBP's *weak* planning algorithms. For better results we could combine MBP's strong-cyclic and weak planning algorithms sequentially — if strong-cyclic planner returns failure then apply weak planner. A planner hybridized in this manner would be able to handle these improper problems comfortably and will also guarantee reaching a goal if it is possible.

The idea of hybridizing planners is quite general and can be applied to various other settings. We list several planning problems that could make good use of a hybridized planner:

1. Partially Observable Markov Decision Processes (POMDP): The scalability issues in POMDPs are much more pronounced than in MDPs due to the exponential blowup of the continuous belief-state representation. We could hybridize GPT or point-based value iteration based methods [Pineau *et al.*, 2003; Poupart, 2005] with disjunctive planners like MBP, BBSP, or POND [Bertoli *et al.*, 2006; Rintanen, 2005; Bryce *et al.*, 2006] to gain the benefits of both the probabilistic and disjunctive representations.
2. Deterministic Over-subscription Planning: The objective of over-subscription planning problem is to maximize return by achieving as many goals as possible given the resource constraints [Smith, 2004]. The problem may be modeled as a search in the state space extended with the goals already achieved. A heuristic solution to the problem could be to achieve the goals greedily in the order of decreasing returns. We can hybridize the two algorithms to obtain a better than greedy, faster than optimal solution for the problem.
3. Probabilistic Planning with Continuous Resources: A Hybrid AO* algorithm on an abstract state space solves this problem. Here each abstract node consists of the discrete component of the state space and contains the value function for all values of continuous resources [Mausam *et al.*, 2005]. This Hybrid AO* algorithm may be hybridized with an algorithm that solves a simpler problem assuming deterministic resource consumption for each action with the deterministic value being the average of consumption distribution.
4. Concurrent MDP: A concurrent MDP is an MDP with a factored action representation and is used to model probabilistic planning problems with concurrency [Mausam and Weld, 2004]. A concurrent MDP algorithm and the equivalent single action MDP algorithm could be hybridized together to obtain a fast concurrent MDP solver.

6 Conclusions

Our work connects research on probabilistic planning with that on qualitative contingent planning — with exciting results. This paper makes the following contributions:

- We present a novel probabilistic planning algorithm (HYBPLAN) that combines two popular planners in a principled way. By hybridizing an optimal (but slow)

planner (GPT) with a fast (but sub-optimal) planner (MBP) we obtain the best of both worlds.

- We empirically test HYBPLAN on a suite of medium and large problems, finding that it has significantly better anytime properties than RTDP. Given limited resources, it is able to solve much larger problems while still computing high quality solutions. Furthermore, HYBPLAN is competitive with the best planners in the 2006 International Planning Competition.
- Our notion of hybridization is a general one, and we discuss several other planning applications where we believe that the idea will be effective.

Acknowledgments

We thank Blai Bonet for providing the source code of GPT. We thank Doug Aberdeen, Olivier Buffet, Zhengzhu Feng and Sungwook Yoon for providing code for their software. We also thank Paul Beame, Gaurav Chanda, Alessandro Cimatti, Richard Korf, Eric Hansen, Subbarao Kambhampati, Sylvie Thiebaux, and Håkan Younes for very helpful suggestions. Pradeep Shenoy, Ravikiran Sarvadevabhatla and the anonymous reviewers gave useful comments on prior drafts. This work was supported by NSF grant IIS-0307906, ONR grants N00014-02-1-0932, N00014-06-1-0147 and the WRF / TJ Cable Professorship.

References

[Barto *et al.*, 1995] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[Bertoli *et al.*, 2001] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a Model Based Planner. In *ICAI-2001 workshop on Planning under Uncertainty and Incomplete Information*, pages 93–97, 2001.

[Bertoli *et al.*, 2006] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong planning under partial observability. *Artificial Intelligence*, 170:337–384, 2006.

[Bertsekas, 1995] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[Bonet and Geffner, 2003] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS'03*, pages 12–21, 2003.

[Bonet and Geffner, 2005] B. Bonet and H. Geffner. mGPT: A probabilistic planner based on heuristic search. *JAIR*, 24:933, 2005.

[Bryce *et al.*, 2006] D. Bryce, S. Kambhampati, and D. E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006.

[Cimatti *et al.*, 2003] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[Feng and Hansen, 2002] Z. Feng and E. Hansen. Symbolic heuristic search for factored Markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.

[Gomes and Selman, 2001] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[Hansen and Zilberstein, 2001] E. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001.

[Little and Thiebaux, 2006] I. Little and S. Thiebaux. Concurrent probabilistic planning in the graphplan framework. In *ICAPS'06*, 2006.

[Martello and Toth, 1990] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[Mausam and Weld, 2004] Mausam and D. Weld. Solving concurrent Markov decision processes. In *AAAI'04*, 2004.

[Mausam and Weld, 2005] Mausam and D. Weld. Concurrent probabilistic temporal planning. In *ICAPS'05*, pages 120–129, 2005.

[Mausam *et al.*, 2005] Mausam, E. Benazara, R. Brafman, N. Meuleau, and E. Hansen. Planning with continuous resources in stochastic domains. In *IJCAI'05*, page 1244, 2005.

[McMahan *et al.*, 2005] H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML'05*, 2005.

[Pineau *et al.*, 2003] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI'03*, pages 1025–1032, 2003.

[Poupart, 2005] P. Poupart. Exploiting structure to efficiently solve large scale partially observable Markov decision processes. Ph.D. thesis. University of Toronto, 2005.

[Rintanen, 2005] J. Rintanen. Conditional planning in the discrete belief space. In *IJCAI'05*, 2005.

[Smith, 2004] D. E. Smith. Choosing objectives in over-subscription planning. In *ICAPS'04*, page 393, 2004.

[Younes and Simmons, 2004] H. L. S. Younes and R. G. Simmons. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS'04*, page 325, 2004.