Improving Determinization in Hindsight for On-line Probabilistic Planning

Sungwook Yoon[1] and **Wheeler Ruml**[2] and **J. Benton**[3] and **Minh B. Do**[1]

[1]Embedded Reasoning Area
Palo Alto Research Center
Palo Alto, CA 94304 USA
sungwook.yoon at parc.com
minhdo at parc.com

[2]Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
ruml at cs.unh.edu

[3]Dept. of Computer Science and Eng.
Arizona State University
Tempe, AZ 85287 USA
j.benton at asu.edu

Abstract

Recently, 'determinization in hindsight' has enjoyed surprising success in on-line probabilistic planning. This technique evaluates the actions available in the current state by using non-probabilistic planning in deterministic approximations of the original domain. Although the approach has proven itself effective in many challenging domains, it is computationally very expensive. In this paper, we present three significant improvements to help mitigate this expense. First, we use a method for detecting potentially useful actions, allowing us to avoid estimating the values of unnecessary ones. Second, we exploit determinism in the domain by reusing relevant plans rather than computing new ones. Third, we improve action evaluation by increasing the chance that at least one deterministic plan reaches a goal. Taken together, these improvements allow determinization in hindsight to scale significantly better on large or mostly-deterministic problems.

Introduction

The unexpected success of FF-Replan (Yoon, Fern, and Givan 2007) in the 2004 International Probabilistic Planning Competition (IPPC) has led to great interest in the use of deterministic planning techniques to solve on-line probabilistic problems (Mausam, Bertoli, and Weld 2007; Yoon et al. 2008). In the latest IPPC in 2008, many competing planners, including the winner, used a deterministic planner in some manner. Rather than precomputing a policy, FF-Replan selects an action by performing planning when it reaches a new state. It first creates a deterministic version of the probabilistic domain model in which action outcomes are assumed to be known, then uses a fast deterministic planner, specifically FF (Hoffmann and Nebel 2001), to find a plan from the current state using the determinized model. It issues the first action in the resulting plan and waits to observe the resulting state. If FF-Replan re-visits a state, it repeats the previously selected action. By planning on-the-fly when unseen states are observed, FF-Replan can often cope with stochastic state transitions despite using a deterministic planner at each step, especially when the problem is dead-end free. However, the approach discards probability information completely and it is not hard to construct domains that exploit its weaknesses (Little and Thiebaux 2007).

Yoon et al. (2008) re-interpreted this method as a degenerate form of 'determinization in hindsight', a principled approach to probabilistic planning and scheduling (Chong, Givan, and Chang 2000; Mercier and Hentenryck 2007). As we explain in more detail later, the hindsight approach samples multiple possible determinizations of the probabilistic planning domain model, representing different possible action outcomes. The deterministic planner is then used to find a plan for each determinization and the set of resulting plans is analyzed to select the action that leads to the best outcome, according to the plans for the sampled futures. Yoon et al. showed that their system, FF-Hindsight, could exhibit significant probabilistic reasoning, solving problems even in 'probabilistically interesting' domains designed to exploit the weaknesses of determinizing planners. However, the approach is very computationally expensive. Specifically, the amount of computation needed at each state scales linearly in the number of actions applicable at that state and in the number of determinizations sampled, leading to enormous wasted effort in domains with many applicable-but-useless actions and when actions have few probabilistic effects.

In this paper, we present three enhancements to the hindsight planning approach that collectively improve its scalability significantly. First, we break the linear dependence on the number of applicable actions by detecting a small set of *probabilistically helpful actions* from the current state. We consider only those actions, estimating their expected cost while skipping all other applicable actions. This idea was originally termed *consensus and expectation* by Bent and Van Hentenryck (2004), who used it in the context of stochastic scheduling. We discuss the properties of this technique, which is also similar to the 'helpful actions' of the FF planner (Hoffmann and Nebel 2001). Second, we show how to exploit determinism in the domain by reusing relevant solution plans. When a state transition happens as was predicted by previously sampled determinizations, those samples and their corresponding plans can be directly reused. This avoids the need to replan when the state evolves according to the anticipated trajectory. In effect, the algorithm gracefully adapts its planning effort to the determinism of the domain. Third, we improve estimation accuracy in hard instances, where most determinizations fail to reach a goal, by supplementing the sampled determinizations with all-outcome determinization, where every probabilistic outcome is incarnated as a possible action. This provides the deterministic planner a chance to reach the goal, regardless of how unlikely the path is. By mixing this all-outcome determinization into the regular determinizations, we promote a minimal level of search guidance. This is particularly use-

ful when there are goals that can only be achieved by the low-probability outcomes of actions that are unlikely to be selected under ordinary sampling.

After discussing these techniques in more detail, we present an empirical evaluation of our approach on domains from previous International Probabilistic Planning Competitions (IPPC). Our improved determinzation in hindsight planner, FF-Hindsight+, scales significantly better than the original FF-Hindsight system, and is competitive with the winner of each IPPC on its respective benchmarks.

Background

We formalize a probabilistic planning problem as a Markov decision process (MDP) $M = \{S, s_0, A, T, C, G\}$, where S is a finite set of states, $s_0 \in S$ is the initial state, A is a finite set of actions, T defines transition probabilities, C is the cost function, and G is a set of goal states. T defines the distribution of the next state s' after applying an action $a \in A$ to s, written $T(s'|s, a)$. The cost function $C : S \times A \to \mathbb{R}$ assigns a real valued cost to a state/action pair. Goal states are sink states—that is, there is no action that leads out of them—effectively terminating the on-line trial. This is the setting used in the International Probabilistic Planning Competition (IPPC) (Younes and Littman 2004; Bonet and Givan 2006).

The IPPC takes an on-line simulation approach to evaluate competing planners. The competition host sends a state s to the participant, with the very first one being the initial state s_0. The participant then returns an action a. The host simulates the application of a in s, choosing a next state s' according to $T(s'|s, a)$, and then initiates a new *turn* by sending s' to the participant. The interaction continues until the participant succeeds in reaching a goal $s' \in G$ or the predefined time expires, which is typically set at 30 CPU minutes. One such complete session is called a *round*. Multiple rounds (typically 30 in the IPPC) are used to compute the average performance of a participant on the same MDP. The number of successful rounds and the total time taken are criteria used to compare the participating planners.

Simulation-based testing makes it reasonable to adopt an on-line planning approach, finding actions as they are needed rather than solving the entire MDP as traditional algorithms do. Surprisingly, many IPPC competitors use deterministic planners as key components of their systems. This is motivated by the huge advantages in speed of deterministic planners over traditional probabilistic planners. The simplest way to use deterministic planners in probabilistic planning is replanning, which only plans when unseen states are visited (Yoon, Fern, and Givan 2007). Replanning first "determinizes" the probabilistic domain, either by taking one outcome of an action as the single deterministic outcome, or by making each probabilistic outcome its own deterministic action, known as 'all outcome determinization.' Either method of determinization discards probability information completely. While FF-Replan was the top performer in IPPC-04, it can perform poorly in simple "probabilistically interesting" problems (Little and Thiebaux 2007).

1. $\mathcal{F} \leftarrow$ sample *num-samples* possible futures to *horizon*
2. for each action a applicable in the current state s
3.   for each possible resulting state s'
4.   for each sampled future $F \in \mathcal{F}$
5.   plan in F from s' using deterministic methods
6.   average costs obtained in plans from s'
7.   compute expected cost of a from values of s'
8. return action with smallest expected cost

Figure 1: Planning via determinization in hindsight.

Determinization in Hindsight (DH)

Determinization in hindsight (DH) takes a more sophisticated approach. Rather than naively determinizing probabilistic outcomes, DH samples them according to their distribution. In this way, it incorporates probabilistic information. It then solves the "sampled" determinized problem with off-the-shelf deterministic planner, achieving scalability. We now explain this technique in more detail.

When executing a probabilistic action, nature 'rolls the dice' to determine the resulting state. In a determinization, we literally roll the dice in advance, recording the output of a pseudo-random generator for H steps, where H is a fixed horizon. More formally, a possible future F is a mapping $A \times S \times \{1, ..., H\} \to [0, 1]$ that forms a determinization by assigning a fixed outcome to each action a in state s at time t by partitioning the interval $[0, 1]$ according to the outcome distribution for action a and then selecting the outcome for a at time t that $F(a, s, t)$ indexes. DH forms a set of these sampled futures where each sample represents a deterministic planning problem that can be solved using a deterministic planner. We can then evaluate a state using the collected set of determinizations. Figure 1 shows the pseudo-code of the basic DH algorithm.

The evaluation of a policy π (mapping from a state to an action) at state s on a future (or determinization) F is $C(s, F, \pi) = \sum_{s \sim F}^{H} C(s, \pi(s))$, which is simply accumulating costs following the state transitions dictated by the future and taking the action specified by π. The optimal value of a state with respect to a limited horizon H is defined by

$$V^*(s) = \min_{\pi} \mathop{\mathrm{E}}_{F}[C(s, F, \pi)], \tag{1}$$

where π is a non-stationary policy for horizon H. Once we can evaluate the values of all states as in Equation 1, we can effectively execute an optimal policy with a one step lookahead procedure. One step lookahead selects the action that leads to the best expected value across the possible resulting states. So the optimal policy π_{V^*} for a state s according to V^* is, $\pi_{V^*}(s) = \mathrm{argmin}_a[C(s, a) + \sum_{s'} V^*(s')T(s'|s, a)]$. The problem with this general approach is that it is computationally very expensive to calculate the value function as defined in Equation 1, since we have to enumerate all the possible policies and evaluate them over the distribution of all the possible futures.

DH alleviates this burden by interchanging expectation and minimization in Equation 1:

$$V^{Hind}(s) = \mathop{\mathrm{E}}_{F}[\min_{\pi} C(s, F, \pi)] \tag{2}$$

Now, the minimization in Equation 2 is actually a deterministic problem, since it is with respect to the already decided future F. $V^{Hind}(s)$ may not be the same as $V^*(s)$ but it can give an approximation of it (Yoon et al. 2008; Mercier and Hentenryck 2007). It may provide an over-optimistic value estimate however, as it assumes that the outcome of each action is known. Use of a satisficing deterministic planner also introduces approximation. To see the gain in computational efficiency, consider the approximation of Equation 2 with w sampled futures:

$$V^{\sim Hind}(s) = \frac{\sum_{1 \leq i \leq w} [\min_\pi C(s, F_i, \pi)]}{w} \quad (3)$$

The efficiency of DH comes from considering w deterministic problems in evaluating actions in the current state. Given recent advances in deterministic planning techniques, $\min_\pi C(s, F, \pi)$ can be computed relatively quickly in practice. Thus, $V^{\sim Hind}(s)$ provides great computational savings.

Improving Determinization in Hindsight

The original DH technique has several potential drawbacks: 1) it evaluates all actions that are applicable in the current state, even those that do not contribute to achieving the goal; 2) it forms many samples even when actions are deterministic; and 3) it performs poorly in problems where goal achievement depends on very unlikely outcomes. We will address each of these issues in turn.

1. Probabilistically Helpful Actions (PHA)

The intuition behind PHA stems from the observation that we might not need to consider all the applicable actions in a given state because only a small subset of them might be useful. Moreover, even if that subset is large, we may not need to consider all of them to find a good one. To illustrate this point, consider a simple domain we call 'fumbling gripper.' It is similar to the classic 'gripper' domain, as well as the 'slippery gripper' domain of Kushmerick, Hanks, and Weld (1995). There are two rooms, A and B, with a robot with a gripper and n balls initially located in room A. The robot needs to move a subset of the balls into room B. The robot can pick up or put down a ball and move between the two rooms. The only complication is that some balls are slippery, and picking them up only succeeds half of the time. If there are no slippery balls, this domain is deterministic, and if all the balls are slippery, then it is highly probabilistic. The percentage of slippery balls to be moved controls how stochastic the problem is.

When the robot is in room A with k balls and the goal is to have only `ball-X` in room B, the optimal plan will begin with picking up `ball-X`. Although the actions of picking up the other $k-1$ balls are also applicable, they are not useful. Conventional DH would consider all applicable actions, evaluating the possible resulting states using the sampled futures. But ideally, we would evaluate only those actions that start an optimal plan. To find these *useful* actions, we evaluate the current state, rather than the one step lookahead states. The first actions in these plans are then considered further using the standard DH algorithm. In addition, note that if the goal is to have both `ball-X` and `ball-Y` in B,
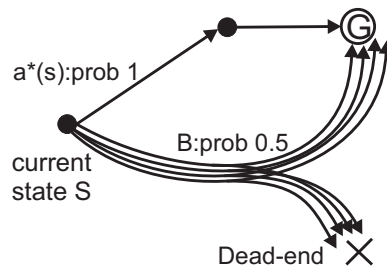

Figure 2: A difficult problem for the PHA technique: $a^*(s)$ is optimal but any action in set B that reaches the goal in a given determinization will appear better.

then the symmetry in the problem implies that we only need to consider moving one ball first. PHA lets us rely on the tie-breaking in the deterministic planner and only consider the action it selects first.

More precisely, before we apply DH, we evaluate the current state s to extract a set $A^H(s)$ of probabilistically helpful actions (PHA). Then in DH, we only evaluate states resulting from the application of actions in $A^H(s)$ to s. Specifically, when evaluating the current state, we form w determinizations $\mathcal{F} = \{F_1, \ldots, F_w\}$, with an extended horizon $H + 1$ starting from the current state (extended since H is measured from the resulting states *after* expanding the current state). We solve these futures using a deterministic planner to obtain a plan set $\mathcal{P}(s) = \{P_1, \ldots P_w\}$, where each plan $P_i \in \mathcal{P}(s)$ is a sequence of actions, $\{a_1^i, \ldots, a_{H+1}^i\}$. The set of the first actions $A^H(s) = \{a_1^1, \ldots a_1^w\}$ of plans in $\mathcal{P}(s)$ is identified as 'probabilistically helpful actions' (PHA) for state s and is used to reduce the number of future states to consider in the subsequent step.

Properties of PHA: We are not guaranteed that $A^H(s)$ will contain the optimal action for s. This is the result of dealing with known futures. Consider the example in Figure 2. In this example, in the current state s, there is one optimal action $a^*(s)$ that reaches the goal with probability 1 but requires an additional step, and there are many bad actions, labeled B, that reach the goal in one step with probability 0.5 but, if they miss, lead to a dead-end state. Clearly, $a^*(s)$ is the optimal action. But if there are many actions from set B available in state s, then PHA will not include $a^*(s)$. To see this, first note that in any determinization, each action in set B will deterministically reach either the goal or the dead-end. If there are N actions in B, the probability that all of them reach the dead-end equals $(\frac{1}{2})^N$. Only in such an unlikely case will action $a^*(s)$ appear in a deterministic solution and become a member of PHA. Otherwise, the planner will never consider it because it takes longer to reach the goal. And as N increases, PHA becomes increasingly vulnerable and will only contain actions from B. We pay this price due to the interchange of expectation and minimization.

On the other hand, there is more to like about PHA than just its efficiency. This comes from the closeness between the $V^{Hind}(s)$ and $V^{\sim Hind}(s)$ equations that we discussed above. If we let the allowed loss of DH using PHA be

ϵ, the Chernoff bound states that with probability $1 - \delta'$, $|V^{Hind}(s) - V^{\sim Hind}(s)| < \sqrt{\frac{-\log \delta'}{w}}$. By setting $\delta' = \frac{\delta}{H+1}$ and $|V^{Hind}(s) - V^{\sim Hind}(s)| < \sqrt{\frac{-\log \delta'}{w}} < \frac{\epsilon}{4}$, we can guarantee that if we sample $w > \frac{16}{\epsilon}^2 \log \frac{H+1}{\delta}$ times, then with probability $1 - \delta$, actions $a^1_{P_1} \ldots a^1_{P_w}$ will not make subsequent plans deviate from $V^{Hind}(s)$ by more than $\frac{\epsilon}{2}$. Combining this bound with the bound described in Theorem 2 of Yoon et al. (2008), which states that the loss of DH can be bounded by $\frac{\epsilon}{2}$ if the number of samples w is $> \frac{16}{\epsilon^2} \log \frac{|A^H|H}{\delta}$, we can bound the total loss of PHA and DH at ϵ, with $w = \frac{16}{\epsilon}^2 \log \frac{H+1}{\delta} + \frac{16}{\epsilon^2} \log \frac{|A^H|H}{\delta}$.

For a given state s, let the number of applicable actions be n, the number of useful actions be $m \le n$, planning time per sampled future be t_p, sampling time per sample be t_F, and the number of sampled futures be w, then the time savings is roughly $(n - (m+1)) \times w \times (t_p + t_F)$. When $n \gg m$, the effect is dramatic and as we will show later, an empirical evaluation confirms that it is frequently obtained in practice.

2. Finding Deterministic Action Sequences

We turn now to our second improvement to DH. In many probabilistic domains, a significant portion of the plan can be deterministic. That is, a sequence of actions that needs to be taken may not have any probabilistic effects. Therefore, identifying and memorizing those sequences can save computation by skipping repeated action evaluations at each intermediate state. For example, in our fumbling gripper example domain, after picking up balls from room A (which may involve slippery stochastic actions), it is deterministic to carry them to room B, drop them, and return to room A. Another example is the *Boxworld* domain of IPPC-04, a variation of the popular *Logistics* domain, in which only the `drive` action is probabilistic.

We can identify deterministic action sequences by comparing, over all sampled futures, the solution plans that start with the action a that has been selected for execution. We find the longest prefix that is common to all plans in that set, record that prefix, and then continuously apply actions from it in sequence without further planning until an action is not applicable. (This could easily be generalized to reuse matching plans, without requiring complete agreement among all samples.) While this is similar to the detection and application of sequences of applicable actions in deterministic planning (c.f., Vidal (2004)), it is perhaps more principled since we can bound the loss with enough samples using an argument similar to that discussed in the previous section.

3. Find Low-Probability but Important Outcome

While surprisingly powerful, sampling-based evaluation techniques are notoriously weak when unlikely outcomes contribute disproportionately to an action's value. This property causes high variation in hindsight evaluation due to the heterogeneity of the sampling results. When an unlikely but game-changing outcome F is sampled using naive sampling, the evaluation result can be significantly different from the evaluation of other sample sets that do not contain F. Techniques such as importance sampling and quasi-

Monte-Carlo evaluation have been introduced to combat this variance (Glasserman 2003). Importance sampling goes beyond naive sampling to sample according to the *probability × cost* distribution instead of the *probability* distribution. This is easier said than done, particularly for planning applications, where it is not easy to identify such distributions from the PDDL domain and problem definitions.

In some of the IPPC benchmark domains such as *Zenotravel*, we observed unlikely outcomes heavily affecting goal achievability. In *Zenotravel*, actions such as 'complete-zooming' have low success probability but the optimal policy involves executing this action repeatedly. For DH to correctly evaluate visited states in such problems, it needs to include this very unlikely outcome in its sample set.

Rather than trying to analyze the domain structure to identify useful but unlikely outcomes specifically, we form a special determinization in which each possible action outcome is instantiated as a separate action. This method was termed all-outcome determinization by Yoon, Fern, and Givan (2007). Planning in this action space essentially gives the deterministic planner complete control over all probabilistic outcomes. If a path to a goal exists, however unlikely, it is a possible solution to the deterministic problem and we would expect the deterministic planner to find it. We then mix in the solution of the all-outcome determinization with our regular DH solutions to futures obtained from naive sampling in our hindsight state evaluations. In practice, this turns out to be highly effective in planning problems where unlikely outcomes greatly affect goal achievability.

Note that when mixing in the all-outcome solution, we also re-weight the solutions for the regular DH sampled futures. Let $P(F)$ be the probability of future F happening, which is the multiplication of the probabilities of outcomes involved with F. When we add in the future F_0 found with the all-outcome determinization to the set of all futures, we re-weight the solutions for each future F by adjusting: $W(F) = \frac{P(F)}{\sum_{F, F_o} P(F)}$. This tempers the optimism inherent in all-outcome determinization while still providing an informative sample in difficult problems.

Putting It All Together

Figure 3 presents the pseudo-code for an improved determinization in hindsight algorithm that includes the three improvements described in the previous sections: (i) probabilistically helpful actions (PHA), (ii) deterministic action sequence detection, and (iii) discovering low-probability but important outcomes.

The algorithm starts by checking whether there is a macro action a_M in progress from deterministic action sequence detection (line 1). If so, and if a_M is still applicable, we apply it (line 2). Since the sequence is identified as a common action sequence among solutions of many futures, it is almost certain that a_M is applicable. Otherwise, we perform PHA from the current state (lines 3-5) and identify useful actions for further evaluation using one step lookahead (line 6). Note that here we mix in the all-outcome solution and its future to the sample future set. We then conduct normal DH with actions identified in line 5.

We reuse the sampled futures from line 3 when possible. If the solution for a sampled future starts with the action

1. if *deterministic-sequence* is not empty
2.   remove its first action, and if it's applicable, return it
3. $\mathcal{F} \leftarrow$ *num-samples* possible futures to *horizon*+1
4. for each future $F \in \mathcal{F}$, find a plan p_F
5. mix all-outcome solution p_{F_o} and its future F_o into $\mathcal{F}$
6. $A^H \leftarrow$ union of the first actions in p_F
7. for each useful action $a \in A^H$
8.   for each future F
9.   if p_F starts with a then reuse p_F
10.   else use deterministic planner to find
            p_F starting with a
11.   compute $Q(s, a)$ using the re-weighted plans
12. find action a^* with the lowest $Q(s, a)$ value
13. check for a *deterministic-sequence* in the plans for a^*
14. return a^* and its *deterministic-sequence*, if any

Figure 3: Improved determinization in hindsight.

being considered in line 7, we use in line 9 the solution previously found, otherwise we resort to deterministic planning to find a new solution p_F for the future F with the constraint that p_F should start with a. After the solutions for all w samples are found, we use them to update the Q-value $Q(s, a)$ in line 11. $Q(s, a)$ is the re-weighted average evaluation of plans following action a. The final action selection in line 12 is the same as in the original DH algorithm: we take the action a minimizing $Q(s, a)$. In the case of the IPPC, since there is no intermediate cost or reward, we use the number of goal achievements of the plans as the primary criterion, breaking ties on plan length. In line 13, we also find the longest common-prefix-plan among all the solution plans for all w futures following a^*, to enable deterministic action sequence detection. Finally, we return the best action and its deterministic macro action sequence, if any.

Empirical Evaluation

We implemented the DH improvements discussed above in a planner called FF-Hindsight+, henceforth abbreviated as FF-H+. Following the FF-Hindsight system of Yoon et al. (2008), henceforth abbreviated as FF-H, we used FF (Hoffmann and Nebel 2001) as the deterministic planner. Both FF-H+ and FF-H generate one random number for each time step when sampling futures; this number is used across all states and actions for that particular time step (see (Yoon et al. 2008) for details and a potential pitfall). We conducted experimental evaluations on the IPPC-04, IPPC-06 and IPPC-08 domains (Younes and Littman 2004; Bonet and Givan 2006) as well as the 'probabilistically interesting' domains introduced by Little and Thiebaux (2007). For each problem, 30 rounds were conducted with a total time bound of 30 CPU minutes for all rounds on that problem and there is no additional limit on the time per turn. Experiments were run on a T9400 2.4 GHz Intel dual-core Linux PC. We used a fixed number of 20 sampled futures across all problems and a horizon of 200. Three planners FF-H+, FF-H, and FF-Replan [1] were run on this setting. For the

---

Domain	FF-H+	FF-H	FF-Replan
bw-c-pc-nr-8	30 (2)	30 (5)	30 (1)
bw-c-pc-8	30 (2)	30 (5)	30 (1)
bw-nc-pc-5	30 (1)	30 (2)	30 (0)
bw-nc-pc-nr-8	30 (2)	30 (5)	30 (0)
bw-nc-pc-8	30 (2)	30 (5)	30 (0)
bw-nc-pc-11	30 (20)	8 (30)	30 (1)
bw-nc-pc-15	0 (-)	0 (-)	0 (-)
bw-nc-pc-18	0 (-)	0 (-)	0 (-)
bw-nc-pc-21	1 (30)	0 (-)	30 (19)
bx-c10-b10-pc-nr	30 (10)	10 (30)	30 (3)
bx-c10-b10-pc	30 (10)	10 (30)	30 (2)
bx-c15-b10-pc	30 (20)	20 (30)	30 (3)
bx-c5-b10-pc-nr	30 (2)	30 (5)	30 (1)
bx-c5-b10-pc	30 (2)	30 (5)	30 (1)
exploding-block	28 (4)	28 (7)	3 (0)
file-prob-pre	14 (30)	14 (30)	14 (30)
g-tire-problem-	18 (2)	18 (2)	7 (0)
r-tire-problem-	30 (2)	30 (2)	30 (0)
toh-prob-pre	17 (2)	17 (11)	0 (-)
ztravel-1-2	30 (5)	0 (-)	0 (-)
Total	**468**	365	414

Figure 4: The number of successful rounds on the IPPC-04 benchmarks, with time used for the successful rounds in minutes in parentheses.
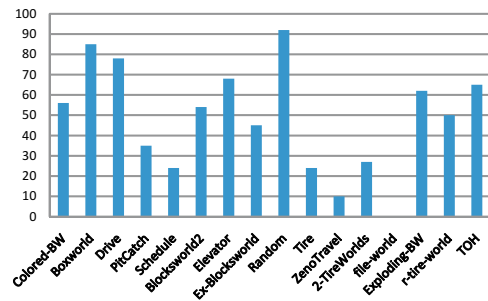

Figure 5: Percentage of actions pruned with PHA technique

other competition winners such as FPG and RFF, given that we were not able to obtain and run them ourselves, data are collected from the official IPPC results. We outline the settings used in those competitions in the respective sections.

Each comparison table in Figure 4, 6, and 8 lists the primary metric, the number of solved rounds, and in parentheses the total CPU time (in minutes) used for those successful rounds.

IPPC-04 Domains

Figure 4 shows the experimental results on IPPC-04 domains. FF-Replan was the winner of this competition. As mentioned above, all three planners were run on the same machine with specs mentioned earlier. Each 'domain' in the IPPC-04 represents a single problem. Names starting with "bw" are *blocksworld* domains varying in colors and number

---

of blocks. The goal of this blocksworld domain is to build a tower with specific colors, rather than a tower with specific blocks. Problems starting with "bx" belong to *boxworld*, a *Logistics* variant. For "bw" domains, FF-H+ clearly outperforms FF-H in terms of solving time when all 30 rounds are successful. It also returns more successful rounds on several instances. The reason is apparent when we look at Figure 5, which shows the percentage of applicable actions that are ignored when using PHA. The *Colored-BW* column in the figure shows that FF-H+ evaluates fewer than half of the actions that FF-H considers. Compared to FF-H, FF-H+ is even better in the "bx" domain with more successful rounds in shorter solving time. Figure 5 again points out the reason: in *Boxworld*, more than 80 percent of actions were not evaluated in FF-H+, leading to significant speedup. Nevertheless, both "bw" and "bx" are deadend-free domains that do not require significant probabilistic reasoning and thus FF-Replan outperforms both FF-H and FF-H+.

Exploding-block and *toh-prob-pre* are two domains in IPPC-04 where probabilistic reasoning is critical. For both of these domains, there are safer paths that take more actions to achieve the goal, while shorter paths are unsafe, leading to dead-end states. Although the winner FF-Replan managed to solve a few rounds, this is totally due to luck. For instance, in the exploding-block problem, it may happen that the exploding block does not explode during naive plan execution, resulting in spurious success. FF-H and FF-H+ however managed to find safer action sequence. For example, in the same problem, FF-H and FF-H+ found ways to neutralize the exploding block. In terms of solving time, FF-H+ is again several times faster than FF-H.

ztravel-1-2 is a variant of the Zeno-travel domain. Besides normal actions, it also has actions starting with "complete" (e.g, "start-zooming" and "complete-zooming"). These "complete-" actions have outcomes with extremely small probability but that outcome achieves the "zooming" effect. Successful plans repeat the "complete-" actions and FF-Replan failed completely in this domain due to its inability to select low-probability actions. FF-H also failed completely, since naive sampling cannot find the unlikely outcome. FF-H+, by mixing in the all-outcome solution, could find the solution, showing a clear benefit of this technique.

For the overall comparison across all domains, FF-H+ returns the highest number of successful rounds (468) with about 25% improvement over FF-H (365) and 12% improvement over FF-Replan (414). In general, FF-Replan is faster than both FF-H+ and FF-H. However, given the smaller number of success rounds, it's likely that FF-Replan hits the dead-ends in the wrong paths more frequently.

IPPC-06 Domains

Figure 6 shows the experimental results on IPPC-06 domains. For reference, we listed the results of the winner in that competition, FPG, which we collected from the official results. FPG also uses a deterministic planner as a subroutine. We tried to obtain FPG and run it ourselves but were unsuccessful. Therefore, unlike the previous results, we caution that the machine used for FF-H+ and FF-H is different from the competition machine. The IPPC-08 report indicates that the time allowed in IPPC-06 for each planner to

Domain	FF-H+	FF-H	FPG
Blocksworld2	335 (15)	256 (30)	283 (1)
Drive	221 (25)	98 (27)	252 (1)
Elevator	292 (27)	214 (30)	342 (1)
Ex-Blocksworld	265 (28)	205 (30)	193 (0)
Pitch-Catch	96 (30)	55 (30)	103 (0)
Random	357 (22)	301 (26)	292 (1)
Scheduling	195 (27)	154 (30)	243 (1)
Tire	364 (27)	343 (30)	337 (0)
Zenotravel	310 (27)	0 (-)	121 (0)
Total	**2435**	1626	2166

Figure 6: Total successful rounds on the IPPC-06 domains, with time in minutes in parentheses.
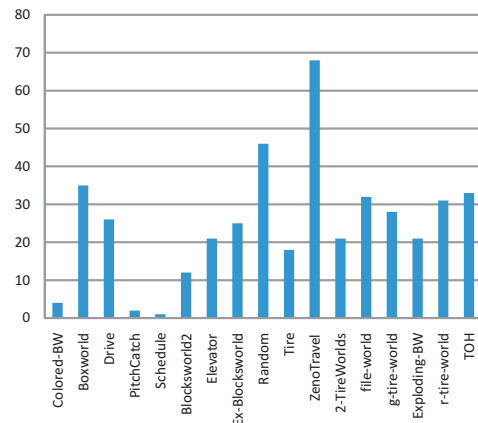

Figure 7: The percentage of actions taken that were chosen using deterministic action sequences.

run in each domain is 40 minutes (we use 30 minutes in our runs). Note also that there are multiple problems (15) for each domain so the maximum total number of successful rounds for each domain is more than 30. The winner FPG was not dominant over all the domains, for example, FOALP performed better on Blocksworld variant. Thus, the numbers are suggestive rather than definitive.

We can see a similar trend on domains that have variations used in IPPC-04, which are *Blocksworld2*, *Ex-Blocksworld*, and *Zenotravel*. Results in *Zenotravel* clearly show the advantage of mixing in all-outcome solution, as FF-H+ is successful in 310 rounds while FF-H failed completely. Deterministic action sequence detection also contributes to the success of FF-H+ in *Zenotravel*, compared to FF-H. Figure 7 shows that FF-H+ took actions from deterministic action sequence close to 70% of the time, saving that proportion from the total computation time.

In Figure 6, *Random* is a randomly generated domain designed to meet STRIPS constraints but with a guarantee that there is a path to the goal in all generated problems. We find that in this domain, there is a large number of useless actions (90%). As can be seen in Figure 5, FF-H+ prunes most of these and performs much better than FF-H (as shown in Figure 6). As we can see from Figure 7, FF-H+ also benefited from using deterministic action sequences. FF-H+ used ac-

Domain	FF-H+	FF-H	RFF
2-TireWorlds	420 (10)	420 (29)	382 (10)
BoxWorld	222 (28)	95 (30)	238 (20)
Blocksworld2	270 (25)	185 (30)	364 (5)
Ex-Blocksworld	214 (23)	131 (30)	58 (1)
Scheduling	251 (30)	221 (30)	99 (1)
Search and Rescue*	450 (7)	450 (27)	0 (-)
SysAdmin-SLP*	0 (-)	0 (-)	117 (0)
Total	**1827**	1502	1258

Figure 8: Total successful rounds on the IPPC-08 domains, with time in minutes in parentheses.

	0%	10%	20%	50%	100%
FF-H+	0	1	1	3	6
FF-H	14	14	14	14	14

Figure 9: Time in minutes to solve 30 successful rounds for the fumbling gripper domain as the percentage of slippery balls varies.

tions from them more than 40% of the time.

The *Pitch-Catch*, *Drive* and *Scheduling* domains are inspired by optimization problems. That is, these domains require sophisticated probabilistic reasoning to maximize the goal reachability probability. FF-H+ and FF-H do not perform well. We believe that they do not have enough sampled futures to differentiate good actions from bad. Automatically adjusting the sample width of DH is an interesting research question, and we believe that adaptive sampling using Hoeffding races could be a useful technique to apply here.

The *Elevator* and *Tire* are domains where simple replanning fails and thus selecting one outcome for an action does not work in these domains. FF-H and FF-H+ both seem to be relatively unaffected by this, rather scalability was the issue for these domains. Overall, FF-H+ manages to solve more problems than FF-H in less time, again showing a clear improvement, as can be evidenced from Figure 5 and 7.

Compared to the FPG, the competition winner, FF-H+ is better in 5/9 domains. Overall, Figure 6 shows that it again returns the highest number of successful round (2435), a 12% improvement over FPG (2166) and 48% improvement over FF-H (1626). In general, FPG is faster than both FF-H+ and FF-H. However, its smaller number of successful rounds suggests that FPG either solves a problem quickly or quickly hits a dead-end and fails.

IPPC-08 Domains

Figure 8 shows results on IPPC-08 domains. For reference, we listed the result of the winner RFF, available from the competition web site. The machine used for the competition was quad core CPU at 2.40 GHz with 4GB of RAM running Linux. While we used uniform 30 minutes cut-off time here, the competition used 40 minutes and occasionally used 10 minutes for smaller problems. Given that the FF-H+, FF-H, and RFF were run on different machines, we again caution that the numbers are not directly comparable but can only be taken as suggestive. However, it is safe to say that the specs on the competition machine is better than ours and the time given is also more generous (40' vs. 30'). Also, note that RFF is not dominant over all domains. For example, there were domains where HMDPP was better.

We also want to note that for the *SysAdmin-SLP* domain, FF-H+ was unable to solve any problem because the syntax complexity caused FF to not be able to solve the determinized version of those problems[2]. We are investigating a

pre-processing approach to compile the *SysAdmin-SLP* domain into a form solvable by FF.

We see the same trend as before for the domains carried over from IPPC-04 and IPPC-06, which are are *Boxworld*, *Blocksworld2* and *Ex-Blocksworld*. For the newly introduced domains, like *2-TireWorlds*, *Scheduling*, and *Search and Rescue*, FF-H+ either solves more problems than FF-H or used less time. The IPPC-08 organizer emphasized "probabilistically interesting" problems with dead-ends and a small likelihood of simple paths, so these problems would likely stymie FF-Replan. The results show that FF-H+ works well in this setting.

Overall, despite the syntactic problem with the *SysAdmin-SLP* domain, FF-H+ still returns the highest number of success rounds (1827), which is 21% better than FF-H, and 45% more than RFF, the competition winner. RFF is generally faster than FF-H+, and FF-H, although this is difficult to interpret due to the different number of successful rounds.

Improvements from PHA

To illustrate the improvement in FF-H+ due to PHA, we show the results of PHA analysis in Figure 5. The bars in this figure show the average percentage of pruned actions among all the available actions in a state. For some domains like *Random* from IPPC-06, the ratio is very high and is close to 90%. Due to space limitations, we use *Colored-BW* to indicate all of the problems in IPPC-04's *blocksworld* and IPPC-06's and IPPC-08's *Blocksworld2* domains.

Bent and Van Hentenryck (2004) show that the PHA technique, which they call "Consensus + Expectation", leads to performance improvements in vehicle routing problem and on-line scheduling problems. The nature of scheduling problems and probabilistic planning problems are quite different, but our results show that a similar benefit can be achieved.

Improvements from Deterministic Sequences

Figure 7 shows the effect of deterministic action sequence detection with the bars show the percentage of actions chosen that way. For most of the domains, the effect is marginally positive except for the *Zenotravel* where the percentage is particularly high, around 70%. In fact, most plans found in the *Zenotravel* domain repeat the "sampled" actions of a deterministic action sequence. For domains like this, following these sequences provides a large computational savings.

To show the behavior of the deterministic action sequence detection technique more concretely, we also tested FF-H+

---

[2]FF-H+ was also not able to solve the original domain formulation for the *Search and Rescue* domain due to the nested syntax. We wrote a preprocessor to flatten this. FF-H+ is then able to solve all problems successfully

Planner	climb	river	tire1	tire10	tire17	Total
FF-H+	30	20	30	30	30	**170**
FF-H	30	20	30	6	0	116
FF-Replan	19	20	15	0	0	54
FPG	30	20	30	0	0	86

Figure 10: Number of successful rounds on probabilistically interesting benchmarks.

on the fumbling gripper domain described earlier in this paper. If there are no slippery balls, then the problem is a deterministic domain and the behavior of an intelligent planner should reflect this. Indeed, our deterministic action sequence detection algorithm can recognize those situations. We tested FF-H+ on fumbling gripper problems with 10 balls with different number of slippery balls. The results of our runs on this domain are displayed in Figure 9. When none of the balls are slippery, FF-H+ acts like a deterministic planner, following deterministic sections of the sampled futures and taking very little planning time. As we increase the number of slippery balls, the computation time of FF-H+ gradually increases.

Probabilistically Interesting Domains

Figure 10 shows the results for the "probabilistically interesting" domains introduced by Little and Thiebaux (2007). We can see that FF-H+ maintains the probabilistic reasoning power of FF-H but scales much better to large problems. For the *Tire World* domains, as the number of tire increases, FF-H+ scales better than FF-H in terms of the amount of solving time. When the number of tires is 10, FF-H completed only 6 rounds while FF-H+ completed all 30 of them. It turns out that FF-H+ can handle the 17 (!) tires problem while none of the other planners can achieve a single successful round. This scalability is particularly encouraging compared to the other planners' steep decline in performance on the 10-tires problem. These results, in concert with those in Figure 4, demonstrate convincingly the advantage of FF-H+ over FF-Replan in this type of problems.

Conclusion

Ironically, deterministic planners have played a significant role in the International Probabilistic Planning Competition. In this paper, we considered one principled technique, determinization in hindsight, and introduced several improvements that greatly increase its scalability. First, we broke its dependence on the number of applicable actions through a technique we called 'Probabilistically Helpful Actions' (PHA). Although we showed that the PHA technique can in theory be misled, in practice it led to improved performance. This technique led to a reduction in computation of up to 90%. Second, we exploited pockets of determinism in a domain by detecting deterministic action sequences. This led to a reduction in computation of up to 67%. Finally, we improve estimation accuracy in problems where it is difficult to reach a goal by mixing in all-outcome determinization. This was crucial to success in several benchmarks. Overall, in every domain tested, we have seen improved performance

over FF-H (Yoon et al. 2008). In some domains, computation time can be orders of magnitude better.

Many of the methods we presented here can be used in a wide variety of problems, especially the use of all-outcome plans. In simulation based evaluation, for example, the concern is the reduction of variance. Importance sampling or Latin hypercube sampling have been developed. However, these techniques require embedding prior human knowledge in the sampling technique. Our all-outcome solutions technique can be viewed as an automated version of importance sampling, and we believe AI planning techniques may well be useful in other simulation problems that need unlikely but important samples to be taken seriously.

References

Bent, R., and Van Hentenryck, P. 2004. The value of consensus in online stochastic scheduling. In *ICAPS*.

Bonet, B., and Givan, R. 2006. International probablistic planning competition. http://www.ldc.usb.ve/~bonet/ipc5/.

Chong, E.; Givan, R.; and Chang, H. 2000. A framework for simulation-based network control via hindsight optimization. In *IEEE CDC Conference*.

Glasserman, P. 2003. *Monte Carlo Methods in Financial Engineering*. Springer.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:263–302.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning 3. *AIJ* 76(1-2).

Little, I., and Thiebaux, S. 2007. Probabilistic planning vs replanning. In *Proceedings of the ICAPS Workshop on the Planning Competitions*.

Mausam; Bertoli, P.; and Weld, D. 2007. A hybridized planner for stochastic domains. In *IJCAI*.

Mercier, L., and Hentenryck, P. V. 2007. Performance analysis of online anticipatory algorithms for large multistage stochastic programs. In *IJCAI*.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *International Conference on Automated Planning and Scheduling*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS*.

Younes, H. L. S., and Littman, M. L. 2004. The first international probabilistic planning competition.