Improved Non-deterministic Planning by Exploiting State Relevance

Christian Muise and **Sheila A. McIlraith**
Dept. of Computer Science
University of Toronto
Toronto, Canada. M5S 3G4
{cjmuise,sheila}@cs.toronto.edu

J. Christopher Beck
Dept. of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada. M5S 3G8
jcb@mie.utoronto.ca

Abstract

We address the problem of computing a policy for fully observable non-deterministic (FOND) planning problems. By focusing on the relevant aspects of the state of the world, we introduce a series of improvements to the previous state of the art and extend the applicability of our planner, PRP, to work in an online setting. The use of state relevance allows our policy to be exponentially more succinct in representing a solution to a FOND problem for some domains. Through the introduction of new techniques for avoiding deadends and determining sufficient validity conditions, PRP has the potential to compute a policy up to several orders of magnitude faster than previous approaches. We also find dramatic improvements over the state of the art in online replanning when we treat suitable probabilistic domains as FOND domains.

1 Introduction

When an agent executes a plan, there may be aspects of the environment over which the agent has no control. One way of modelling this issue is to incorporate non-deterministic actions into the domain model. In the planning community, non-deterministic action outcomes have been introduced in two formalisms: fully observable non-deterministic (FOND) planning (Daniele, Traverso, and Vardi 2000) and probabilistic planning (Yoon, Fern, and Givan 2007). The former deals primarily with finding a contingent plan or policy, while the latter focuses on maximizing the probability of achieving the goal. When considering the probability of successful execution alone, the two formalisms are quite similar and the planners for each share many characteristics.

The state of the art for both FOND planning and probabilistic planning involve *determinizing* the actions by replacing every non-deterministic action with a set of deterministic ones and using a classical planner to find a solution (Fu et al. 2011; Yoon et al. 2010). Both approaches build a partial policy that maps the state of the world to an action and then simulate execution of this policy. When the planner encounters a state that the policy does not recognize, it considers the state as a new planning problem and the planner updates the policy with a newly computed plan. The key difference is the FOND techniques simulate every potential outcome of a non-deterministic action while the probabilistic planning techniques consider only one randomly chosen action outcome for each action in the plan. Both approaches consider information about the entire state, which is often far too detailed. A more succinct policy should map only the relevant portion of the state to the actions.

We develop a planner, PRP, that incorporates both FOND and online probabilistic planning techniques to build a strong cyclic solution to a FOND planning problem. If no such plan exists, PRP returns the best quality policy it is able to find. The strength of our approach stems from focusing on only those parts of the state that are relevant – in many real world problems only a small subset of the state plays a role in the successful execution of a plan. We introduce several novel techniques for non-deterministic planning based on state relevance and demonstrate how these improvements allow our planner to outperform the state-of-the-art techniques in both FOND and probabilistic planning.

PRP generates solutions up to several orders of magnitude faster, and generates policies several orders of magnitude smaller than the state-of-the-art FOND planner, FIP (Fu et al. 2011). When compared to online replanning approaches for probabilistic planning problems, we find that PRP achieves the goal with up to several orders of magnitude fewer actions than the method employed by FF-Replan (Yoon, Fern, and Givan 2007). Further, when we consider "probabilistically interesting" domains that have the potential for deadends (Little and Thiebaux 2007), we find that PRP scales better than the state of the art in online replanning, FF-Hindsight+ (Yoon et al. 2010), solving problems with a perfect success rate where FF-Hindsight+ does not.

2 Preliminaries

We assume that we are given a non-deterministic planning problem as a PDDL file with "oneof" clauses in the action effects, as is the case with FOND planning domains (Bryce and Buffet 2008). We convert the domains to a non-deterministic SAS$^+$ formalism using a modified version of the PDDL-to-SAS$^+$ translation algorithm (Helmert 2009).

We adopt the notation of (Mattmüller et al. 2010) for non-deterministic SAS$^+$ planning problems. A SAS$^+$ *fully observable non-deterministic* (FOND) planning task is a tuple $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A} \rangle$. $\mathcal{V}$ is a finite set of variables v, each having the finite domain D_v. We use D_v^+ to denote the extended domain of v that includes the value $\perp$ which signifies

the undefined state of v. A *partial state* is a function s that maps a variable $v \in \mathcal{V}$ to a value in D_v^+. If $s(v) \neq \perp$ then v is *defined* in s, and if every variable $v \in \mathcal{V}$ is defined for s then s is a *complete state*. The initial state s_0 of a SAS$^+$ FOND planning task is a complete state, while the goal state s_* is a partial state. A partial state s entails another partial state s', denoted as $s \models s'$, iff $s(v) = s'(v)$ whenever v is defined for s'. Two partial states s and s' are said to be *consistent* with one another, denoted $s \approx s'$, iff $\forall v \in \mathcal{V}, s(v) = s'(v) \vee s(v) = \perp \vee s'(v) = \perp$. The *updated* partial state obtained from applying partial state s' to partial state s, denoted as $s \oplus s'$, is the partial state s'' where $s''(v) = s'(v)$ if v is defined for s', and $s''(v) = s(v)$ otherwise. Note that s and s' need not be consistent for $s \oplus s'$.

The final component of a planning task is the set of actions $\mathcal{A}$. Each action is made up of two parts: Pre_a, a partial state that describes the condition under which a may be executed; and Eff_a, a finite set of partial states that describe the possible outcomes of the action. If Eff_a contains multiple elements, then the agent is not able to choose which effect takes place. An action a is *applicable* in state s iff $s \models Pre_a$ and a is *possibly applicable* in s iff $s \approx Pre_a$. The *progression* of a partial state s, w.r.t. an action a and selected nondeterministic effect $e \in \mathit{Eff}_a$, denoted $Prog(s, a, e)$, is the updated state $(s \oplus Pre_a) \oplus e$ when a is (possibly) applicable in s, and undefined otherwise. We say that a partial state s *can be regressed* through a with effect $e \in \mathit{Eff}_a$ iff $e \approx s$. The *regression* of partial state s w.r.t. an action a and effect $e \in \mathit{Eff}_a$, denoted $Regr(s, a, e)$, is the updated s' where $s'(v) = Pre_a(v)$ if v is defined for Pre_a; else $s'(v) = \perp$ if $e(v) = s(v)$; else $s'(v) = s(v)$. $Regr(s, a, e)$ is undefined if s can not be regressed through a with e.

A solution to a FOND planning task is a policy that maps a state to an appropriate action such that the agent eventually reaches the goal. A policy is *closed* if it returns an action for every non-goal state a policy reaches and a state s is said to be *reachable* by a policy if there is a chance that following the policy leads the agent to s. When the agent executes an action the effect is randomly chosen, so a closed policy must handle every possible outcome of an action it returns. There are three types of plans for a FOND problem (Daniele, Traverso, and Vardi 2000): *weak*, *strong*, and *strong cyclic*.

Definition 1 (Weak Plan)**.** A *weak plan* is a policy that achieves the goal with non-zero probability.

A weak plan may be as simple as a sequence of actions that achieves the goal with assumed non-deterministic action outcomes. The policy for a weak plan need not be closed.

Definition 2 (Strong Plan)**.** A *strong plan* is a closed policy that achieves the goal and never visits the same state twice.

A strong plan provides a guarantee on the maximum number of steps to achieve the goal but is often too restrictive.

Definition 3 (Strong Cyclic Plan)**.** A *strong cyclic plan* is a closed policy that achieves the goal and every reachable state can reach the goal using the policy.

A strong cyclic plan guarantees that the agent eventually reaches the goal, but does not guarantee the agent can do so in a fixed number of steps. There are a few approaches to produce a strong cyclic plan and we focus on one that enumerates weak plans until it creates a strong cyclic plan.

Definition 4 (Determinization)**.** A *determinization* of a SAS$^+$ FOND planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A} \rangle$ is a planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{A}' \rangle$ where $\mathcal{A}'$ is a modification of $\mathcal{A}$ such that every action in $\mathcal{A}'$ is deterministic. The *single outcome* determinization creates $\mathcal{A}'$ by selecting a single outcome for every action in $\mathcal{A}$. The *all outcomes* determinization creates $\mathcal{A}'$ by creating a new action for every non-deterministic outcome of an action in $\mathcal{A}$.

Finding a classical plan to a determinization provides a weak plan for the non-deterministic planning task. Recently, (Fu et al. 2011) showed that it is effective to use this approach repeatedly to build a strong cyclic plan. The idea involves picking an unhandled reachable state, finding a weak plan in the all outcomes determinization, and incorporating the plan into the policy. The planner repeats this process until the policy is strong cyclic or it finds a deadend and backtracks to replan. Similar techniques have been used in online replanning for probabilistic planning problems (Yoon, Fern, and Givan 2007; Yoon et al. 2010).

Working Example: Triangle Tireworld To situate the methods we introduce, we use a problem from the triangle tireworld domain as a running example (cf. Figure 1). The objective is to drive from location 11 to 15, however driving from one location to another has the possibility of a tire going flat. If there is a spare tire in a location of the car (marked by circles in the diagram), then the car can use it to fix a flat. The strategy that maximizes the probability of success is to drive to location 51 and then over to 15, as this means a spare tire will always be available.

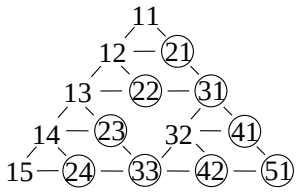

Figure 1: Example from the triangle tireworld domain.

The triangle tireworld problem poses significant difficulty for probabilistic planners (Little and Thiebaux 2007). We also found it to be hard for strong cyclic planners due to the attractive nature of driving straight to the goal as well as the vast number of states that the optimal policy can reach.

3 Finding a Strong Cyclic Plan

To create a strong cyclic plan, we use the all outcomes determinization to find a weak plan when we encounter a state that our policy does not handle. In the presence of deadends, we replan from scratch rather than backtracking. By exploiting state relevance, we make substantial improvements over current methods. In this section, we describe how we build a policy offline, and later describe how we restrict our offline approach to accomplish online replanning. We first describe precisely what a policy is and the notation surrounding it.

Definition 5 (Partial Policy)**.** A *state-action pair* is a tuple $\langle p, a\rangle$ where p is a partial state and a is an action. A *rule set* $\mathcal{R}$ is a set of state-action pairs and $\mathcal{R}(s)$ denotes the set of state-action pairs $\langle p, a\rangle \in \mathcal{R}$ such that $s \models p$. If we have a function Φ to select a desired pair from a rule set, the *partial policy* for $\mathcal{R}$ and Φ is the partial function P that maps a partial state s to the desired element in $\mathcal{R}$: $P(s) = \Phi(\mathcal{R}(s))$. If P is defined for s, we say that P *handles* s.

Given a rule set and a selection function over the possible state-action pair subsets, we can use the partial policy P to map a given state to an appropriate action: the question remains how we build the rule set and selection function. A significant aspect of our solution involves solving the all outcomes determinization for various initial states, creating a weak plan. Our planner then computes a rule set that corresponds to the actions in the plan and the relevant conditions for the plan to succeed. The selection function we use, Φ_{weak}, chooses the desired state-action pair to be the one that is closest to the goal with ties broken arbitrarily. We associate the action outcome from the weak plan to the corresponding non-deterministic action in the state-action pairs.

Traditionally, the state in a state-action pair $\langle s, a\rangle$ corresponds to the complete state s after executing the plan up to just before a. Rather than using the full state, we employ repeated regression from the goal to just before action a in the weak plan. Approaches for execution monitoring use regression to determine precisely which part of the state is relevant in order for the a plan to succeed (Fritz and McIlraith 2007). We use this regressed state as the partial state for the state-action pair and associate the pair with its distance to the goal in the weak plan in which it was found.

We refer to the procedure of creating a weak plan for the all outcomes determinization and then adding the corresponding state-action pairs to the policy P as GENPLANPAIRS(Π, P). Algorithm 1 shows the high-level approach to generating a strong cyclic plan. For now, we only describe aspects of Algorithm 1 that do not involve handling deadends, and we describe the details for deadends below.

Algorithm 1 enumerates the reachable states of the policy by considering every outcome of the actions our policy returns (lines 10-13). The algorithm deals with unhandled reachable states by finding a weak plan to achieve the goal and adding the state-action pairs to the policy (lines 8-9).

One difference between Algorithm 1 and FIP is the outer loop of line 2. We must repeat the process until we find a fixed point because the use of partial states can cause previously handled states to have a new action returned by the policy. If the policy behaves differently, it has the chance of leading to a new unhandled open state. We refer to a single execution of lines 2-13 as a *pass*. The check on line 2 succeeds whenever the previous pass has augmented the policy or a deadend was found. Similar to FIP, GENPLANPAIRS ceases planning when it finds a state handled by the policy.

Theorem 1. If a policy generated by Algorithm 1 handles state s, then following the policy along with the expected action outcomes leads the agent to the goal in finite steps.

Proof sketch. Every state-action pair $\langle p, a\rangle$ in the policy is a part of a weak plan that achieves the goal. Executing the action with the expected outcome brings the state to a point where the next action (denoted a') in the weak plan is applicable. It may happen that the agent selects the next state-action pair from another weak plan, but this pair must be no further from the goal than a' was (since the policy uses Φ_{weak} for comparison). With this invariant on the state-action pairs, we are guaranteed to reach the goal and not encounter an unhandled, deadend, or repeated state. □

The monotonic decrease in the distance-to-goal means that once GENPLANPAIRS reaches a state that our policy handles, we can always follow the policy and expected action outcomes to arrive at the goal and generate a weak plan.

Algorithm 1: Generate Strong Cyclic Plan

```
   Input: FOND planning task Π = ⟨V, s_0, s_*, A⟩
   Output: Partial policy P
1  Initialize policy P
2  while P changes do
3      Open = {s_0}; Seen = {};
4      while Open ≠ ∅ do
5          s = Open.pop();
6          if s ⊭ s_* ∧ s ∉ Seen then
7              Seen.add(s);
8              if P(s) is undefined then
9                  GenPlanPairs(⟨V, s, s_*, A⟩, P);
10             if P(s) is defined then
11                 ⟨p, a⟩ = P(s) ;
12                 for e ∈ Eff_a do
13                     Open.add(Prog(s, a, e));
14     ProcessDeadends();
15 return P;
```

3.1 Avoiding Deadends

The FIP planner handles deadends by backtracking through the policy updates and searching for a new weak plan that avoids the deadend state. Issues with this approach include failing to recognize the same deadend later on in the search and failing to recognize the core reason for the deadend. We now describe how our approach, while incomplete at finding all deadends, addresses these two issues to sufficiently provide a complete search for a strong cyclic policy.

Definition 6 (Forbidden State-Action Pair)**.** We define a *deadend* for the SAS$^+$ FOND planning task $\langle \mathcal{V}, s_0, s_*, \mathcal{A}\rangle$ to be a partial state p such that $\forall s \in S, s \models p$, there is no strong cyclic plan for the task $\langle \mathcal{V}, s, s_*, \mathcal{A}\rangle$. We define a *forbidden state-action pair* to be a state-action pair $\langle p, a\rangle$ such that $\exists e \in \mathit{Eff}_a, s \models Prog(p, a, e)$ where s is a deadend.

In other words, a forbidden state-action pair $\langle p, a\rangle$ is a situation where executing the action a in any state that entails p could lead us to a deadend state. Note that we use deadend to refer to partial states with no strong cyclic plan.

Rather than backtracking, we record the deadends during one pass of Algorithm 1 and avoid them in all subsequent passes. A deadend may arise for two reasons: (1) GENPLANPAIRS fails to find a solution in the all outcomes de-

terminization or (2) during the search for a weak plan, the planner discovers a deadend state. We record all instances of both types of deadends and process them in the final step of a pass (PROCESSDEADENDS in Algorithm 1).

The first step of PROCESSDEADENDS is to *generalize* the deadends: for each deadend s we compute a minimal partial state p such that $p \approx s$ and p suffices to be a deadend in the delete relaxation of the all outcomes determinization (detected using reachability analysis). A generalized deadend may be the entire state, saving nothing, but often we generalize a deadend to a much smaller partial state.

Next, we generate a rule set of forbidden state-action pairs by regressing the deadends through every non-deterministic action effect for every action. The rule set captures the ways in which a non-deterministic action execution may fail. Once we generate the rule set for every deadend recorded in a pass, we add all of the forbidden state-action pairs to a global rule set used during planning. We require only that a single non-deterministic effect lead to a deadend state for the action to appear in a forbidden state-action pair.

The final step of PROCESSDEADENDS is to reset the policy so that we can start the computation over with the knowledge of forbidden state-action pairs. For GENPLANPAIRS, we modified a forward-search planner to restrict the expansion of nodes in the search frontier to avoid forbidden state-action pairs by filtering the actions applicable at every point in the search. We also modified the heuristic computation to account for the forbidden state-action pairs.

In the triangle tireworld example, a common deadend found by our planner is for the car to be in a non-goal location with a flat tire and no spare. The generalization of this deadend removes all information pertaining to the location of spare tires in other locations. The forbidden state-action pairs created for this deadend include any state that has the car one location away without a flat and the action that drives the car to the deadend location. The forbidden state-action pairs allow the GENPLANPAIRS procedure to avoid driving to the problematic location.

Theorem 2. Algorithm 1 computes a strong cyclic plan for a FOND problem if such a plan exists.

Proof sketch. For soundness and completeness, we need only look at the final iteration of the outer while-loop. In the final pass, the policy does not change and no new deadend is found. Therefore, the condition on line 8 can only be met if $s = s_0$, in which case no strong cyclic plan exists. Otherwise, we know that the policy handles s_0 and we can follow the policy to eventually achieve the goal for some arrangement of non-deterministic action outcomes (due to Theorem 1). Further, since the condition on line 8 always fails, the policy always returns an action for any state we reach. Since we start with the initial state in the open list (line 3) and enumerate every possible successor for the non-deterministic actions chosen by the policy (lines 12-13), we are guaranteed that the policy represents a strong cyclic plan. Finally, we know that there is a final pass of Algorithm 1 since the policy monotonically adds further state-action pairs to cover unhandled states (line 9) and forbidden state-action pairs to handle discovered deadends. □

4 Extensions

The techniques presented so far are sufficient to produce a strong cyclic plan if one exists. In this section, we present two extensions. The first is an extension of a technique introduced in the FIP planner. The second is a novel technique for determining if a policy is a strong cyclic plan for a state.

4.1 Planning Locally

One of the significant contributions of FIP was to investigate an unhandled state in the search for a strong cyclic plan in a special way: a local plan to get back to the intended state was sought prior to replanning for the goal (stopping early if the policy handles a state in the search space). We leverage reasoning about partial states to make planning locally more efficient. Rather than planning for the complete state that we expected our planner to be in, we plan for the expected partial state. Formally, if at state s, $P(s) = \langle p, a \rangle$, the expected action outcome of a is $e \in \mathit{Eff}_a$ and we must consider an unexpected action outcome $e' \in \mathit{Eff}_a$. Planning locally occurs when $P(Prog(s, a, e'))$ is undefined. Instead of searching for a plan from $Prog(s, a, e')$ to $Prog(s, a, e)$, we search for a plan from $Prog(s, a, e')$ to $P(Prog(s, a, e))$.

The partial state we plan for contains only the relevant portion of the intended complete state, which is often more succinct and easier to achieve. In the running example, if the agent drives to a location with a spare with the expected outcome of not getting a flat tire, it must handle the case where a flat tire does occur. The complete expected state includes not having a flat tire and a spare tire being present at the destination – a state we can no longer achieve when driving causes a flat. However, the partial expected state does not require a spare tire to exist at the destination, but only that we do not have a flat tire.

When we plan locally we do not record deadends as a strong cyclic plan that achieves the goal may still exist. If planning locally fails, the approach is no different than before since the planner records nothing and we subsequently plan for the goal. If planning locally succeeds, it reaches a state that our policy handles and by Theorem 1 we retain soundness and completeness. The technique can provide a large advantage in certain domains, but we discovered that when we scale other domains, planning locally was detrimental to performance. Planning locally tends to only provide a benefit when the local plan is extremely short. As such, we limit the search effort when planning locally.

4.2 Strong Cyclic Confirmation

Up to this point we completely enumerate every state reachable by a policy to certify the policy is strong cyclic. We improve on this exhaustive approach by identifying states where our policy properly acts as a strong cyclic plan.

Definition 7 (Strong Cyclic State-action Pairs)**.** We define a *strong cyclic state-action pair*, with respect to a policy P, to be a state-action pair $\langle p, a \rangle$ such that for any state s where $P(s) = \langle p, a \rangle$, P is a strong cyclic plan for the state s.

If we knew precisely which state-action pairs in our policy were strong cyclic, then we could stop expanding states whenever we arrive at one that corresponds to a strong cyclic

Algorithm 2: Mark State-Action Pairs

Input: Planning problem Π and rule set $\mathcal{R}$
Output: Annotated rule set $\mathcal{R}$

1 $\forall \langle p, a\rangle \in \mathcal{R}, \langle p, a\rangle.marked = True$;
2 **while** *Some pair becomes unmarked* **do**
3 **foreach** $\langle p, a\rangle \in \mathcal{R}$ *s.t.* $\langle p, a\rangle.marked \wedge p \not\models s_*$ **do**
4 **foreach** $e \in \mathit{Eff}_a$ **do**
5 **if** $\nexists \langle p', a'\rangle \in \mathcal{R}$ *s.t.* $p' \models Prog(p, a, e)$ **then**
6 $\langle p, a\rangle.marked = False$;
7 **else if** $\exists \langle p', a'\rangle \in \mathcal{R}$ *s.t.*
8 $\langle p', a'\rangle.marked = False \wedge$
9 $p' \approx Prog(p, a, e)$ **then**
10 $\langle p, a\rangle.marked = False$;
11 **return** $\mathcal{R}$;

state-action pair. The condition for a pair to be a strong cyclic, however, is difficult to compute. We instead determine sufficient conditions for a state-action pair to be strong cyclic. Algorithm 2 outlines how we compute pairs that match this condition, denoted as being *marked*.

Algorithm 2 works by first assuming that every state-action pair is marked, and then iteratively un-marking those that no longer satisfy the condition to remain marked. The algorithm repeats until it reaches a fixed point, and returns the resulting rule set. Lines 4-10 of the algorithm determine if the state-action pair $\langle p, a\rangle$ should remain marked and there are two conditions that can cause a pair to become unmarked; if either condition holds for an effect $e \in \mathit{Eff}_a$, then we unmark the pair. The first condition ensures that if the policy returns $\langle p, a\rangle$ for a state s, there is at least one state-action pair returned by the policy in the state $Prog(s, a, e)$ (line 5). We require this so that if the policy uses $\langle p, a\rangle$, then it handles every possible outcome. The second condition ensures that if the policy returns $\langle p, a\rangle$ for state s, every state-action pair returned by the policy in a state reached by executing a is itself marked (lines 7-9). The condition may be overly zealous in checking possible state-action pairs. A better approximation of the applicable pairs has the potential to leave more pairs marked and is left as future work.

Theorem 3. If Algorithm 2 leaves a state-action pair of policy P marked, then it is strong cyclic with respect to P.

Proof sketch. Assume for the sake of contradiction that Algorithm 2 completes and there exists a marked state-action pair $\langle p, a\rangle$ that is *not* a strong cyclic state-action pair. Since $\langle p, a\rangle$ remained marked, we know that for any state s if $P(s) = \langle p, a\rangle$, then $\forall e \in \mathit{Eff}_a, P(Prog(s, a, e))$ is defined and further, $P(Prog(s, a, e))$ must be marked. We also know from Theorem 1 that $P(Prog(s, a, e))$ is a pair closer to the goal than $\langle p, a\rangle$. Inductively, we extend this reasoning until we arrive at goal states which are trivially marked. The collective set of states that we reach serve as a certificate for P to be a strong cyclic plan for state s, and thus violates our assumption. □

We incorporate marked state-action pairs into Algorithm 1 by adding a condition on line 10 – we only expand the outcomes of an action in a state-action pair if the pair is not marked. The marking of state-action pairs in the rule set for a policy is only valid until we modify the policy. As soon as we introduce a new state-action pair into the policy, we recompute the marking from scratch. After line 9 in Algorithm 1, we run Algorithm 2 to compute a valid marking.

To take advantage of the marked state-action pairs, we modify the search conducted by Algorithm 1. Line 5 of this algorithm selects an arbitrary state in the set of open states. FIP explores the states in a breadth first manner, but we do not require this for the algorithm to be sound and complete. We instead explore the states in a depth first manner, investigating open states that are closer to the goal in weak plans that we find. The exploration works in concert with Algorithm 2 to progressively mark more of the state-action pairs.

One final change we make to increase the efficiency of using Algorithm 2 is to strengthen the conditions in the state-action pairs of our policy. If the policy returns the state-action pair $\langle p, a\rangle$ for state s with the intended effect $e \in \mathit{Eff}_a$, and we find that a new plan must be computed for the unintended effect $e' \in \mathit{Eff}_a \backslash e$, then we have a newly created state-action pair $\langle p', a'\rangle$ where $P(Prog(s, a, e')) = \langle p', a'\rangle$. We incorporate the condition for the new plan to succeed into p by replacing $\langle p, a\rangle$ with $\langle p \oplus Regr(p', a, e'), a\rangle$. Since the applicability of a non-deterministic outcome depends only on the precondition of the action, we know that p and $Regr(p', a, e')$ are consistent (i.e., $p \approx Regr(p', a, e')$).

Strengthening the partial states has the consequence of making the partial states in the state-action pairs less general, but the benefits outweigh the drawback. Doing so allows the condition on line 5 of Algorithm 2 to fail much more frequently, allowing more states to remain marked. The intuition behind this modification is to strengthen the condition in a state-action pair to include the relevant part of the state for all possible outcomes to succeed.

For the triangle tireworld example, we find state-action pairs where the partial state contains only the location of the car and the fact that the tire is not flat. Strengthening the state-action pair adds the status of spare tires the car needs in the future to the partial state and allows the marking of Algorithm 2 to be much more effective.

Theorem 4. If modified to plan locally and use Algorithm 2, Algorithm 1 computes a strong cyclic plan if one exists.

Proof sketch. We have already seen that planning locally does not affect the soundness and completeness of Algorithm 1. From Theorem 3 we know that any marked state-action pair is a strong cyclic state-action pair. Looking at the final pass of Algorithm 1 that uses all extensions, we can see that the policy remains unchanged, as does the marking of the state-action pairs in the policy. The search through reachable states would therefore follow the policy's choices, or stop when a marked state-action pair signifies that the policy is a strong cyclic plan for the reached state. □

5 Offline vs Online

Our approach to building a policy repeatedly discards the previous policy in the presence of deadends, and then generates a new one that avoids the deadends. While the policies

are not strong cyclic, they may still be of use. We evaluate the quality of a policy by simulating nature and observing how often the policy achieves the goal. If the offline approach does not find a strong cyclic plan, either because one does not exist or because the planner has run out of time, it returns the policy with the highest quality found.

When the planner is executing actions from the policy, it behaves similarly to online replanning approaches for probabilistic planning. If the planner encounters a state that it does not recognize, it computes a weak plan using GENPLANPAIRS and updates the policy accordingly. When replanning online we only augment the current policy. The monotonic nature of our policy construction means that we must ignore forbidden state-action pairs.

By allowing our planner to process the problem offline, we are able to take advantage of finding a policy that maximizes the probability of success. Not taking deadends into account is one of the major drawbacks of the online replanning approach of FF-Replan (Yoon, Fern, and Givan 2007), and FF-Hindsight addressed this issue by sampling several potential futures before deciding on the action it should take (Yoon et al. 2008; 2010). The philosophy we take is to do this simulation prior to execution, allowing us to avoid potentially disastrous outcomes and build a more robust policy.

6 Evaluation

We implemented our approach and extensions to FOND planning by augmenting Fast Downward (FD) (Helmert 2006). In our planner, PRP, we altered many aspects of FD to make it suitable for non-deterministic planning, but the specific details are outside the scope of this paper.

To ascertain the effectiveness of our approach, we evaluate our planner's efficiency on a range of domains from the FOND and probabilistic planning benchmark suites. Whenever possible (and appropriate), we compared PRP to FIP. As our planner is not tailored to use the competition software from the probabilistic planning track, we created a special version of PRP to compare with techniques employed by FF-Replan. All experiments were conducted on a Linux desktop with a 2.5GHz processor, and we limited the execution of the planners to 30 minutes and 2GB memory.

As in previous work, the run times reported do not include pre-processing of the domain. For PRP, the pre-processing time was typically a fraction of a second unless the problem size became prohibitively large. There are many components to PRP, however planning for a weak plan in the determinized domain represents the majority of time our planner takes to find a solution: usually less than 1% of the time is spent in other parts of the overall planning process.

To evaluate the efficiency of finding a strong cyclic plan we assess the time it takes to compute the plan (in seconds). For the quality, we adopt the standard of using the number of state-action pairs in the policy (smaller being better). When evaluating the online replanning efficiency of our approach, we consider the mean number of actions needed to reach the goal, the mean number of replans needed, the mean time it takes to reach the goal, and the number of successful trials when executing the plan with our computed policy.

Domain	No. of Problems	FIP Solved (unsat)	PRP Solved (unsat)
blocks	30	**30** (**0**)	**30** (**0**)
faults	55	**55** (**0**)	**55** (**0**)
first	100	**100** (**25**)	**100** (**25**)
forest	90	20 (11)	**66** (**48**)
blocks-new	50	33 (**0**)	**46** (**0**)
forest-new	90	51 (**0**)	**81** (**0**)
Total	415	289 (36)	**378** (**73**)

Table 1: The number of problems for which FIP and PRP either successfully find a strong cyclic plan or prove none exists (the latter is shown in brackets).

6.1 Offline Planning Efficiency

To measure the efficiency of PRP at computing a strong cyclic plan, we first consider the benchmark suite from the FOND track of the 2008 International Planning Competition (IPC). The suite contains four domains: blocksworld (blocks), faults, first responders (first), and forest. Not every problem has a strong cyclic plan. The top of Table 1 details the number of problems each planner solved, along with the number of those found to not have a solution.

PRP has a distinct advantage in the FOND benchmark domains, but both planners are capable of solving the majority of the problems quickly. In the forest domain, where this is not the case, many problems do not have a strong cyclic plan. Both PRP and FIP produce the optimal policy for problems in the faults and first domains – search for a weak plan in these domains goes directly towards the goal, and planning locally always successfully repairs the policy. In the forest domain, the only version of FIP that we had access to, falsely identifies 11 domains as having a strong cyclic plan when none exists (these do not appear in the table).

To investigate how the planners scale in the two domains that were not trivially handled, we generated a larger benchmark set for the blocksworld and forest domains. For the forest domain, we modified the problem generator slightly to guarantee that every problem has a strong cyclic plan. For the blocksworld domain, we scaled the number of blocks from 1 to 50 (the highest number of blocks in the original benchmark set is 20). We show the coverage for these domains at the bottom of Table 1. For the problems both solved, Figures 2a and 2b show the relative time to compute and size of a policy respectively. Figure 2c shows the coverage for each planner as a function of time. We found that PRP significantly outperformed FIP in these domains, improving both size and run time by an order of magnitude.

Impact of Relevance To demonstrate an extreme example where state relevance plays a vital role in computing a policy, we introduce the concept of irrelevant action outcomes.

Definition 8 (Irrelevance)**.** We define an *irrelevant variable* $v \in \mathcal{V}$ such that $s_*(v) = \bot$ and $\forall a \in \mathcal{A}, Pre_a(v) = \bot$. That is, v is never relevant for the applicability of an action or the goal. We define an *irrelevant action outcome* e_i to be an outcome of action a that differs from another out-

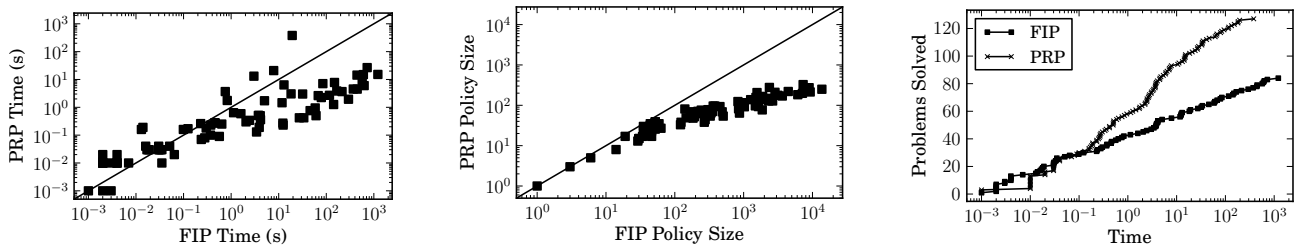

Figure 2: Time to find a strong cyclic plan, and size of the plan's policy for PRP and FIP in the newly generated blocksworld and forest benchmark sets. Figure 2c shows the number of problems solved if given a limited amount of time per problem.

come of a by only irrelevant variables: $\exists e \in \mathit{Eff}_a \setminus e_i, \forall v \in \mathcal{V}, e_i(v) = e(v)$ or $e_i(v) = \bot$ or v is irrelevant.

The presence of irrelevant action outcomes causes a planner that uses full state to spend unnecessary time generating a strong cyclic plan. In the extreme, the performance of a planner using the full state is exponentially worse.

Theorem 5. In a domain with irrelevant action outcomes, our approach computes a policy exponentially smaller in the number of irrelevant action outcomes than the smallest policy computed by existing methods that use the entire state.

Proof sketch. Consider the case in which every action randomly changes a set of irrelevant variables. The number of irrelevant action outcomes is exponential in the number of irrelevant variables and a strong cyclic plan must be able to handle every one of them. Using only the relevant portion of the state means that our approach only needs to find a plan for the relevant action outcomes. The states reached by irrelevant action outcomes match a partial state in our policy, while approaches using complete states must replan. □

To investigate the empirical impact of state relevance, we modified the original blocksworld benchmark domain to have a varying number of irrelevant variables that every action non-deterministically flips to true or false. Due to limitations of FIP on the number of non-deterministic effects, we compare a version of our planner that uses complete state for everything (PRP_{Full}) to a version that uses partial states only for the policy it constructs (PRP). The degree of irrelevance, k, was varied between 1 and 5 variables creating up to a factor of $2^5 = 32$ times the number of non-deterministic outcomes for each action. Out of 150 problems, using the full state solves only 85 problems while using the partial state solves 130 problems – the remaining 20 problems have too many ground actions to even be parsed. Figures 3a, 3b, and 3c show the time and size comparison for all k. We find a clear exponential separation between the two approaches and for higher k the difference in time to find a solution and size of the policy is up to several orders of magnitude.

Probabilistic Planning Domains For domains with probabilistic action outcomes, we ignore the probabilities and treat the problem as a FOND planning task. We investigated three domains from the IPC-2006 probabilistic planning track: blocksworld-2 (bw-2), elevators (elev), and zeno-travel (zeno). We also considered the four "probabilistically interesting" domains introduced in (Little and Thiebaux 2007): climber, river, bus-fare, and triangle-tireworld (tire).

Domain (# Probs)	Success Rate (%) FF-H+	Success Rate (%) PRP	Total Time (sec.) FF-H+	Total Time (sec.) PRP
bw-2 (15)	74.4	**100**	900	**8.4**
elev (15)	64.9	**100**	1620	**1.7**
zeno (15)	68.9	**100**	1620	**98.7**
climber (1)	**100**	**100**	-	0
river (1)	**66.7**	**66.7**	-	0
bus-fare (1)	**100**	**100**	-	0
tire-1 (1)	**100**	**100**	-	0
tire-17 (1)	**100**	**100**	-	18.5
tire-35 (1)	-	**100**	-	1519.5

Table 2: Percentage of successful runs out of 30 per problem for PRP and FF-Hindsight+ in a range of domains.

In Table 2 we show the percentage of successful runs (out of 30 per problem) and the total time spent computing a policy for all of the problems in a domain. For a rough comparison, we include the results for FF-Hindsight+ (FF-H+) reported in (Yoon et al. 2010) ('-' indicates unreported results).

While our experimental setup is not identical to the probabilistic IPC track, the efficiency of PRP in finding a strong cyclic plan in these domains is a strong indication that processing the planning problems before execution has a great deal of promise. Most notable is the ability of PRP to solve large triangle tireworld problems. Previously, FF-Hindsight+ claimed the record for this domain by solving tire-17, but through the advances introduced in this paper PRP is capable of solving tire-35 in under 30 minutes.

6.2 Online Replanning Efficiency

We measure the efficiency of our approach to compute plans online and replan in the presence of unexpected states. We give PRP zero time to process the domain before it starts execution and compare ourselves to PRP_{R}: a version of our planner that behaves like FF-Replan, using complete states rather than partial ones. Similar to the offline case, we see little improvement in the standard benchmark set as they are all fairly trivial for online replanning (with the notable exception of the forest domain where deadends are quite common and online replanning fails consistently). We also found this to be the case for the selection of probabilistic planning problems that yield strong cyclic plans.

We instead present results on the generated blocksworld (bw-new) and forest (fr-new) benchmarks described earlier,

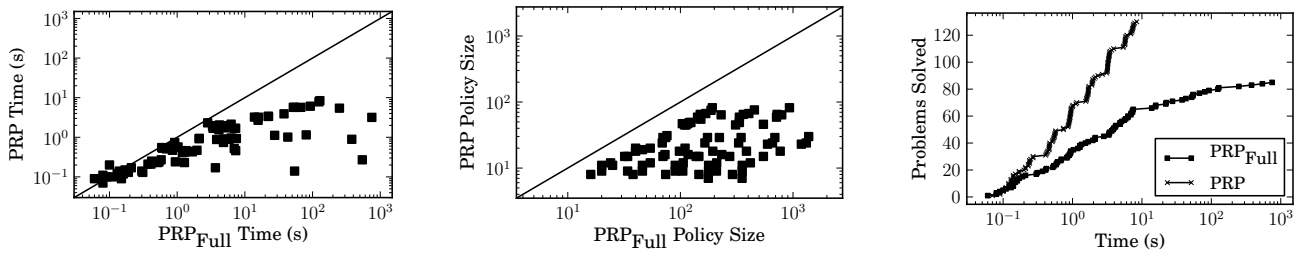

Figure 3: Time to find a strong cyclic plan, and size of the plan's policy for PRP and PRP$_{\text{Full}}$ in the blocksworld domain with irrelevant fluents added. Figure (c) shows the number of problems solved if given a limited amount of time per problem.

Domain	Avg. Actions		Avg. Replans		Avg. Time	
	PRP$_{\text{R}}$	PRP	PRP$_{\text{R}}$	PRP	PRP$_{\text{R}}$	PRP
bw-new	77.4	**75.8**	**5.3**	**5.3**	3.2	**2.6**
fr-new	**60.4**	65.7	3.9	**3.5**	0.6	**0.2**
bw-irr-2	17593.3	**73.6**	67.2	**3.0**	3.4	**1.2**

Table 3: Mean number of actions, replans, and time (in seconds) for PRP and PRP$_{\text{R}}$ to reach the goal.

as well as the blocksworld domain with two irrelevant fluents added (bw-irr-2). Domains with further irrelevant fluents caused PRP$_{\text{R}}$ to not complete many of the problems. Table 3 presents the average number of actions, states requiring replanning, and time to achieve the goal.

We observe an increase in performance on the newly generated benchmarks, but most notably a striking decrease in the number of actions required in a domain with irrelevant fluents. The variance for the number of replans and time to achieve the goal is fairly low for PRP$_{\text{R}}$, but the number of actions required is extremely high in a handful of problems. The large increase is due to the planner making poor choices when replanning for the goal and ending up in a cycle of states that it has low probability of escaping. There is always a non-zero chance of escaping, which is why both PRP and PRP$_{\text{R}}$ have perfect coverage in the tested domains.

7 Related Work

Directly related to our work are the approaches for computing strong cyclic plans. FIP is the most recent and effective solver for FOND problems (Fu et al. 2011). (Mattmüller et al. 2010) propose an LAO* search based on pattern database heuristics to solve SAS$^+$ FOND planning tasks and older approaches use either general game playing or symbolic model checking (Kissmann and Edelkamp 2009; Cimatti et al. 2003). Algorithm 2 is also related to the strong extension phase presented in (Cimatti et al. 2003). The key difference is that we only search through the state-action pairs in our policy when marking; not all possible pairs in the domain. Additionally, our approach may return a strong cyclic policy even if a strong policy exists.

In online replanning, similar approaches include FF-Replan and FF-Hindsight+ (Yoon, Fern, and Givan 2007; Yoon et al. 2008; 2010). FF-Replan replans online for a new weak plan when the agent encounters an unrecognised state. FF-Hindsight+ samples several futures by planning for multiple time-dependent determinizations. The sampling allows FF-Hindsight+ to avoid bad areas of the state space if enough futures fail when a bad action choice is taken.

Our use of relevance to construct a policy is inspired by approaches for execution monitoring (Fritz and McIlraith 2007; Muise, McIlraith, and Beck 2011). They use regression to determine relevant portions of the state to build a policy for online execution. Similar approaches have been proposed for probabilistic planning to compute a so-called *basis function* for the existence of a weak plan that has a non-zero probability of reaching the goal (Sanner 2006; Kolobov, Mausam, and Weld 2010a). The deadend generalization we employ is also related to computation of deadends in (Kolobov, Mausam, and Weld 2010b); the key difference is how our initial candidate is chosen. Finally, the condition strengthening we use for strong cyclic detection is related to the regression of a formula through a non-deterministic action to determine relevance (Rintanen 2008).

8 Conclusion

We exploit state relevance to achieve state-of-the-art performance in non-deterministic planning. Our planner is capable of finding a strong cyclic plan for a FOND problem several orders of magnitude faster than the previous state of the art and generates a policy representation that can be exponentially smaller. Key contributions include a new approach for handling the presence of deadends in a non-deterministic domain and a novel method to compute sufficient conditions for a policy to be a strong cyclic plan. The improvements we make to computing a policy for planning in the presence of non-deterministic action outcomes represent a significant step forward in planner capabilities for this type of problem.

In the future, we hope to focus on improving the reasoning in domains where action outcome probabilities are known by following an approach similar to (Bidot 2005). We also plan on developing better techniques for reasoning with deadends in non-deterministic domains. Finally, we intend to investigate the reason behind the success of planning locally. Continuously trying to get back to the main plan may result in a policy that is less robust or has a higher expected plan length.

Acknowledgements We gratefully acknowledge funding from the Ontario Ministry of Innovation and the Natural Sciences and Engineering Research Council of Canada.

References

Bidot, J. 2005. *A General Framework Integrating Techniques for Scheduling under Uncertainty*. Ph.D. Dissertation, École Nationale d'Ingénieurs de Tarbes, France.

Bryce, D., and Buffet, O. 2008. 6th International Planning Competition: Uncertainty Track. In *Proceedings of International Planning Competition*.

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

Daniele, M.; Traverso, P.; and Vardi, M. Y. 2000. Strong cyclic planning revisited. *Recent Advances in AI Planning* 35–48.

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 144–151.

Fu, J.; Ng, V.; Bastani, F. B.; and Yen, I.-L. 2011. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In *Proceedings of the 22nd International Joint Conference On Artificial Intelligence (IJCAI)*, 1949–1954.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173(5-6):503–535.

Kissmann, P., and Edelkamp, S. 2009. Solving fully-observable non-deterministic planning problems via translation into a general game. In *Proceedings of the 32nd Annual German Conference on Advances in Artificial Intelligence*, KI'09, 1–8. Berlin, Heidelberg: Springer-Verlag.

Kolobov, A.; Mausam; and Weld, D. S. 2010a. Classical planning in MDP heuristics: with a little help from generalization. In *20th International Conference on Automated Planning and Scheduling (ICAPS)*, 97–104.

Kolobov, A.; Mausam; and Weld, D. S. 2010b. SixthSense: Fast and reliable recognition of dead ends in MDPs. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 1108–1114.

Little, I., and Thiebaux, S. 2007. Probabilistic planning vs replanning. *ICAPS Workshop International Planning Competition: Past, Present and Future*.

Mattmüller, R.; Ortlieb, M.; Helmert, M.; and Bercher, P. 2010. Pattern database heuristics for fully observable nondeterministic planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 105–112.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2011. Monitoring the execution of partial-order plans via regression. In *Proceedings of the International Joint Conference On Artificial Intelligence (IJCAI)*, 1975–1982.

Rintanen, J. 2008. Regression for classical and nondeterministic planning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 568–572.

Sanner, S. 2006. Practical linear value approximation techniques for first-order MDPs. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 1010–1016.

Yoon, S.; Ruml, W.; Benton, J.; and Do, M. B. 2010. Improving determinization in hindsight for online probabilistic planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*, 209–216.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 352–359.