Monitoring the Execution of Partial-Order Plans via Regression

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

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

Abstract

Partial-order plans (POPs) have the capacity to compactly represent numerous distinct plan linearizations and as a consequence are inherently robust. We exploit this robustness to do effective execution monitoring. We characterize the conditions under which a POP remains viable as the regression of the goal through the structure of a POP. We then develop a method for POP execution monitoring via a structured policy, expressed as an ordered algebraic decision diagram. The policy encompasses both state evaluation and action selection, enabling an agent to seamlessly switch between POP linearizations to accommodate unexpected changes during execution. We demonstrate the effectiveness of our approach by comparing it empirically and analytically to a standard technique for execution monitoring of sequential plans. On standard benchmark planning domains, our approach is 2 to 17 times faster and up to 2.5 times more robust than comparable monitoring of a sequential plan. On POPs that have few ordering constraints among actions, our approach is significantly more robust, with the ability to continue executing in up to an exponential number of additional states.

1 Introduction

Partial-order plans (POPs) reflect a least commitment strategy [Weld, 1994]. Unlike a sequential plan that specifies a set of actions and a total order over those actions, a POP only specifies those action orderings necessary to the achievement of the goal. In so doing, a POP embodies a family of sequential plans – a set of linearizations all sharing the same actions, but differing with respect to the order of those actions.

Partial-order planning has been less popular in recent years in great part because of the speed with which sequential planners can produce a solution using heuristic search techniques. However we argue that it is not enough to generate plans quickly; an agent or agents must also be able to successfully execute a plan to achieve their goal, and often must do so in the face of an unpredictable and changing environment. For many problems, POPs, when combined with effective execution monitoring, can provide flexibility and robustness when it's needed most – at execution time. To investigate this claim, we examine the problem of POP execution monitoring.

Models of the world and the effects of agents' actions on the world are often imperfect, leading to changes in the state of the world that deviate from those predicted by our models. In execution monitoring (EM), the state of the world is monitored as a plan is being executed. When there is a discrepancy between the predicted and observed world state, a typical EM system attempts to repair the plan or replan from scratch. EM may have many stages including state estimation, evaluating if execution can continue (state evaluation), selecting the action to perform (action selection), and replanning in the presence of plan failure. In this work we are primarily concerned with *state evaluation* and *action selection*: given a plan and the current state of the world, can we continue executing our plan, and if so how should we proceed.

Effective EM systems determine the subset of *relevant* conditions that preserve plan validity and focus on discrepancies with respect to these conditions. Shakey the robot's triangle tables were an attempt to model such conditions [Fikes *et al.*, 1972]. Recently Fritz and McIlraith [Fritz and McIlraith, 2007] characterized the conditions under which a partially executed sequential plan remains valid as the regression of the goal back through the plan that remains. An EM algorithm compares the conditions associated with each step of the plan to the state of the world and proceeds with the plan from the matching point. For partial-order planning, EM is perhaps best exemplified by the SIPE [Wilkins, 1985] and Prodigy [Veloso *et al.*, 1998] systems which take a different approach: monitoring the violation of so-called *causal links* and attempting to repair them as necessary.

In this paper we address the problem of POP EM by building on the insights developed for sequential plan EM. We likewise appeal to a notion of regression to identify the conditions under which a POP remains viable. We then compile these conditions into a structured policy that compactly maps them to an appropriate action. We evaluate our approach analytically and empirically, comparing it to the standard technique for monitoring sequential plans. On International Planning Competition (IPC) domains where there are numerous ordering constraints, and as such few distinct POP linearizations, our approach is 2 to 17 times faster and up to 2.5 times more robust. On POPs that have few ordering constraints, the number of states for which our POP remains viable may be

exponentially larger than the number of such states resulting from a sequential plan. We identify further characteristics of a POP that give our approach a distinct advantage.

In the next section we provide a brief description of related approaches, the state of the art in execution monitoring, and describe our characterization of POP viability. We describe our approach to EM using a structured policy in Section 3 and present the experimental results in Section 4. We conclude with a discussion in Section 5.

2 Execution Monitoring of a POP

In this paper we restrict our attention to STRIPS planning problems. A planning problem is a tuple $\Pi = \langle F, O, I, G\rangle$ where F is a finite set of facts, O is the set of operators, $I \subseteq F$ is the initial state, and $G \subseteq F$ is the goal state. A *complete state* (or just state) s is a subset of F. Facts not in s are interpreted as being false in the state. An operator $o \in O$ is defined by three sets: $PRE(o)$, the facts that must be true in order for o to be executable; $ADD(o)$, the facts that operator o adds to the state; and $DEL(o)$, the facts that operator o deletes from the state. An *action* refers to a specific instance of an operator, and we say that an action a is *executable* in state s iff $PRE(a) \subseteq s$. Similarly, a sequence of actions $\vec{a}$ is executable, if the preconditions of each action in the sequence are true in the corresponding state. We say that a state entails a formula, $s \models \psi$, if the conjunction of facts in s with the negation of the facts not in s logically entails the formula ψ. If ψ is a conjunction of positive facts then $s \models \psi$ iff the conjuncts of ψ form a subset of s. A sequential plan for Π is a sequence of actions $\vec{a}$ such that $\vec{a}$ is executable from I and achieves G. We refer to the *suffix* of a sequential plan (or sequence of actions) $\vec{a} = [a_1, \ldots, a_n]$ to be the empty sequence or the sequence of actions $[a_i, \ldots, a_n]$ where $i \geq 1$. We define the *prefix* of a sequential plan analogously.

An EM system typically monitors the execution of a plan with the objective of ensuring that the plan is executing as intended. When something goes awry, the system takes an ameliorative action such as repairing the plan or replanning from scratch. Here we address the problem of monitoring the execution of a POP, with a view to exploiting its inherent flexibility and robustness.

We define a POP with respect to a planning problem Π as a tuple $\langle \mathcal{A}, \mathcal{O}\rangle$ where $\mathcal{A}$ is the set of actions in the plan and $\mathcal{O}$ is a set of orderings between the actions in $\mathcal{A}$ (e.g., for $a_1, a_2 \in \mathcal{A}, (a_1 \prec a_2) \in \mathcal{O}$) [Weld, 1994]. A total ordering of the actions in $\mathcal{A}$ that respects $\mathcal{O}$ is a *linearization*. A POP provides a compact representation for multiple linearizations.

Depending on how the POP was constructed, it may include a set of causal links. Each causal link contains a pair of ordered actions and a fact that the first action achieves for the second. Causal links often serve as justifications for the ordering constraints. We do not exploit them in our approach to EM, but previous systems for POP EM, such as SIPE [Wilkins, 1985] and Prodigy [Veloso *et al.*, 1998], typically monitor and exploit causal links. If a causal link becomes violated, these systems attempt to repair the plan or replan from what is known of the current state. The Prodigy system extended this approach to interleave planning and execution. Both SIPE and Prodigy monitor the validity of the entire POP that remains to be executed. In contrast, we would like to monitor the validity of any potential partial execution of the POP and not only the current partial execution. Doing so allows us to continue executing the POP from an arbitrary point in the plan.

We take an approach to EM that builds directly on insights from EM for sequential planning dating back as far as Shakey the robot [Fikes *et al.*, 1972], and recently formalized by Fritz and McIlraith [2007]. A central task of execution monitoring is to determine whether the plan being executed remains valid, given what is known of the current state. Recall that given a planning problem $\Pi = \langle F, O, I, G\rangle$ a sequential plan is *valid* iff the plan is executable in I and G holds in the resulting state. We extend this definition to say that the sequential plan *remains valid with respect to* s iff there is a suffix of the plan that is executable in s and G holds in the state resulting from executing the suffix in s. It is the validity with respect to the current state that is at the core of monitoring sequential plan execution.

We define the validity of a POP analogously: given a planning problem Π, a POP P is *valid* with respect to state I iff every linearization of P is valid. We could similarly define the notion of a POP remaining valid relative to a state s of the world, but validity is clearly too strong a condition. Rather, given that a POP compactly represents multiple linearizations, an appropriate analogue is to ensure that at least one of these linearizations remains valid.

Definition 1 (POP viability)**.** Given a planning problem Π and associated POP P, P is *viable* with respect to state s iff there exists a linearization of P that remains valid for s.

Whereas the objective of EM for sequential plans is to determine whether a plan remains valid, we claim that the objective of POP EM is to determine if the POP is *viable* with respect to the current state. Following the methodology adopted for EM of sequential plans, we can address this question effectively by identifying the relevant conditions that lead to POP viability and ensure that one of these conditions holds.

Fritz *et al.* formalized such conditions by characterizing them in terms of *regression* [Waldinger, 1977]. For our purposes, we exploit a simple form of regression, restricted to STRIPS, and limit our exposition accordingly. Regression is a form of syntactic rewriting that allows us to compute the weakest condition that must hold prior to the execution of an action in order for a formula to hold after the action occurs. We formally define regression as follows:

Definition 2 (Regression in STRIPS)**.** Given a planning problem Π and a conjunction of facts, ψ, expressed as a set of facts, we define the *regression* of a conjunctive formula ψ with respect to an action a, denoted $\mathcal{R}[\psi, a]$, as follows: $\mathcal{R}[\psi, a] = (\psi \setminus ADD(a)) \cup PRE(a)$, if $ADD(a) \subseteq \psi$ and $DEL(a) \cap \psi = \emptyset$ (otherwise $\mathcal{R}[\psi, a]$ is undefined). The repeated regression over a sequence of actions $\vec{a}$, denoted as $\mathcal{R}^*[\psi, \vec{a}]$, is simply the repeated application of the regression operator through each action in the sequence (assuming it is defined at each step): e.g., if $\vec{a} = [a_1, a_2, a_3]$ then $\mathcal{R}^*[\psi, \vec{a}] = \mathcal{R}[\mathcal{R}[\mathcal{R}[\psi, a_3], a_2], a_1]$.

Exploiting the notion of regression, Fritz *et al.* identified

the conditions that ensure the validity of a plan: given a sequential plan $\vec{a} = [a_1, \ldots, a_n]$, $\vec{a}$ remains valid with respect to a world state s, iff s entails one of the following conditions: $\mathcal{R}[G, a_n]$, $\mathcal{R}^*[G, [a_{n-1}, a_n]]$, $\ldots$, $\mathcal{R}^*[G, \vec{a}]$. These conditions were integrated into an EM algorithm that checked the condition associated with each suffix, from the shortest to the longest suffix (i.e., the original plan) and resumed execution of the first suffix whose associated condition was entailed by the state (Def. 3, [Fritz and McIlraith, 2007]). If no such condition was found, the EM system would decide to replan. We refer to this approach as the *Sequential Method.*

Returning to EM of POPs, since we have defined POP viability in terms of the remaining validity of a POP linearization, it follows that we can define analogous conditions for each POP linearization and the union of these conditions comprise the conditions for POP viability.

Proposition 1. Given a planning problem $\Pi = \langle F, O, I, G \rangle$, a POP P is viable with respect to state s iff at least one linearization of P has a suffix $\vec{a}$ such that $s \models \mathcal{R}^*[G, \vec{a}]$.

As there may be an extremely large number of linearizations, computing the conditions for each one is inefficient. However, there is often structure in a POP that we can exploit to compute the conditions for POP viability more efficiently. To this end, we provide a method for constructing conditions that avoids enumerating all of the linearizations. Intuitively, we regress the goal back through the POP, exploiting the conditions and actions shared amongst the linearizations suffixes. During the process, we gradually reduce the POP until we have enumerated every condition.

To construct the conditions we use the following notation:

- $last(\langle \mathcal{A}, \mathcal{O} \rangle) \stackrel{\text{def}}{=} \{a \mid a \in \mathcal{A} \wedge \nexists\, (a \prec a') \in \mathcal{O}\}$: The set of actions that appear in a POP such that there is no ordering constraint originating from the action.
- $prefix(\langle \mathcal{A}, \mathcal{O} \rangle, a) \stackrel{\text{def}}{=} \langle \mathcal{A} \backslash a, \mathcal{O} - \{(a' \prec a) \mid a' \in \mathcal{A}\} \rangle$ is the POP that remains after we remove action a and all of the associated ordering constraints from the POP. $prefix(\langle \mathcal{A}, \mathcal{O} \rangle, a)$ is undefined if $a \notin last(\langle \mathcal{A}, \mathcal{O} \rangle)$.

Definition 3 (Γ-conditions)**.** A Γ-condition is a tuple containing a formula and a POP. Given a planning problem $\Pi = \langle F, O, I, G \rangle$ and POP $P = \langle \mathcal{A}, \mathcal{O} \rangle$, we define the set of Γ-conditions for P and Π as $\Gamma_{\Pi,P} = \bigcup_{i=0}^{|\mathcal{A}|} \gamma_i$, where $\gamma_0 = \{\langle G, P \rangle\}$ and we define γ_i inductively as follows:

$$\gamma_{i+1} = \bigcup_{\langle \psi, P \rangle \in \gamma_i} \{\langle \mathcal{R}[\psi, a], prefix(P, a) \rangle \mid a \in last(P)\}$$

Intuitively, every tuple in γ_i contains a condition for a linearization suffix of size i to be a valid plan from the current state, as well as a POP that contains the actions not in the suffix. We relate the conditions for POP viability and the formula $\Gamma_{\Pi,P}$ through the following theorem.

Theorem 1 (Condition Correspondence)**.** Given a planning problem Π, the POP P is viable with respect to state s iff $\exists$ $\langle \psi, P' \rangle \in \Gamma_{\Pi,P}$ such that $s \models \psi$.

Proof sketch. Any linearization of the POP P must end with an action found in $last(P)$, otherwise it would violate an ordering constraint. From this, we can see that every set of actions that make up a linearization suffix will be enumerated, and a POP corresponding to the actions not in the suffix will appear in a tuple. Theorem 1 holds for γ_0, and inductively we find that for every level i, γ_i will contain tuples for every set of actions that make up a suffix of size i and their associated conditions, establishing an equivalence between the conditions of $\Gamma_{\Pi,P}$ and the conditions of the linearizations of P. Thus, following Proposition 1, Theorem 1 holds. $\square$

With a method for computing the required conditions for POP viability, we turn our attention to how we exploit these conditions for the overall EM strategy.

2.1 Condition-Action List

To put the conditions for POP viability to use, we must determine what the agent's behavior will be when a condition is met. Below we deal with the case when the current state satisfies more than one condition, but assuming that a condition is met, we ultimately want to return an appropriate action. In the construction of the Γ-conditions, we are continuously choosing the next action through $last(P)$. To build a mapping of conditions to actions, we record the action that was used to construct a condition in an ordered list called the *Condition-Action List.* Our final condition-action list will map a regressed formula to a single action. Using the construction of $\Gamma_{\Pi,P}$, we present a procedure for computing the condition-action list in Algorithm 1.

Algorithm 1: Condition-Action List Generator

Input: POP $\langle \mathcal{A}, \mathcal{O} \rangle$. Planning problem $\Pi = \langle F, O, I, G \rangle$.
Output: List of (ψ, a) pairs.

```
L = [ ];   // L is the list of (ψ, a) pairs to be returned
Γ = {⟨G, ⟨A, O⟩⟩};   // Γ is a set of tuples of the form ⟨ψ, P⟩
for i = 1 ··· |A| do
    foreach ⟨ψ, P⟩ ∈ Γ do
        foreach a ∈ last(P) do
            L.append( (R[ψ, a], a) );
    /* Update to γ_{i+1} */
    Γ = ⋃_{⟨ψ,P⟩∈Γ} {⟨R[ψ, a], prefix(P, a)⟩ | a ∈ last(P)};
return L;
```

The algorithm begins by initializing Γ to contain the entire POP, and the goal as the associated formula. In each iteration, we update Γ and add the (ψ, a) pairs to the list. Note the order of the (ψ, a) pairs in L: if one pair appears after another, we know it must be from a suffix of equal or larger size. The ordering of L is crucial for the next step of our approach, since we prefer to execute actions closer to the goal.

Theorem 2 (Correctness of Algorithm 1)**.** Given a planning problem Π and associated POP P, the tuples returned by Algorithm 1, with input P and Π, are precisely those in $\Gamma_{\Pi,P}$ and the associated actions correspond to the first action in the linearization suffix associated with the condition.

Proof sketch. The conditions computed by Algorithm 1 correspond precisely to those in $\Gamma_{\Pi,P}$ since line 7 performs the update for successive γ_i steps and line 6 adds the conditions for each step. Since the actions chosen in line 5 are from $last(P)$, the actions in the tuples correspond to the first action in the suffix associated with the condition. □

We now have a specification of our objective (POP viability), and an algorithm to compute the conditions under which a POP remains viable (the condition-action list). Next, we look at how to put this information to use.

3 POP Policy

Our approach for execution monitoring of a POP is to generate a *structured policy* that maps states to actions. Given a state, the policy returns the action that gets us as close to the goal as possible. We refer to this procedure as the *POP Method*. By using the POP Method, we avoid the need to check numerous conditions for the current state. We also benefit from having an action returned that gets us as close to the goal as any linearization with the Sequential Method. Our contribution includes how we build, represent, and use the structured policy for execution monitoring.

A structured policy is a function that maps any state to a single action [Boutilier *et al.*, 1995]. We have elected to use an Ordered Algebraic Decision Diagram (OADD) [Bahar *et al.*, 1997] to represent our structured policy. An OADD is a rooted directed acyclic graph where every node is either a leaf node or an inner node. We associate an action (or $\perp$) to every leaf node and a fact to every inner node. Inner nodes have precisely two outgoing edges – a True and False edge corresponding to the truth of the fact.

OADDs have one further restriction: the order of facts from any root to leaf path must follow a predefined order. The order ensures that if we check two facts on a path from the root to a leaf node, we will always check them in the same order. An Ordered Binary Decision Diagram (OBDD) is similar to an OADD, with the main difference being that we associate either True or False to a leaf node and not an action.

Once we have our condition-action list, we embody the following high-level behavior in our policy:

Property 1 (Opportunistic Property)**.** For a state s, define a *valid linear suffix* as a linearization suffix of our POP that is valid with respect to s. If at least one valid linear suffix exists, then return the first action of the shortest valid linear suffix. If more than one qualifies as the shortest, pick one arbitrarily.

We achieve this property as long as the condition-action list is in the correct order. To build the policy, we generate an OADD where the inner nodes correspond to the truth of a fact and the leaf nodes correspond to actions (or $\perp$ when we do not have a matching condition). We found through experimentation that ordering the facts based on where they appear in the condition-action list is highly effective at producing a smaller policy.

To build our policy, we apply the ITE method for OADD's [Bahar *et al.*, 1997] in successive steps. The ITE method takes in two OADD policies (Pol_1, Pol_2), and an OBDD ($obdd$) that all follow the same order. It returns the OADD with the following semantics when evaluating on state s: if $obdd(s)$ holds then return $Pol_1(s)$, otherwise return $Pol_2(s)$.

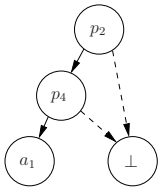


Figure 1: Simple OADD for the pair $(\{p_2, p_4\}, a_1)$

Algorithm 2: POP Policy Generator

Input: Condition-Action List L in sorted order.
Output: Structured policy mapping state to action.

1 $\pi =$ **policy**$(L.pop())$; // π is the current overall policy.
2 **while** $|L| > 0$ **do**
3 $\quad next =$ **policy**$(L.pop())$;
4 $\quad \pi =$ **ITE**(**OBDD**$(next), next, \pi)$;
5 return π;

Two key aspects for building the policy are how we choose the individual policies to begin with, and subsequently how we combine them into one overall policy. We do the former by creating an individual policy for each (ψ, a) pair in our condition-action list L, and we achieve the latter by repeated use of the ITE operation.

To create a policy for each (ψ, a) pair, we need only to follow the pre-defined ordering that respects ψ until it is fully implied in the OADD, and then add a as a leaf node. For example, assume our ordering was $p_1, \ldots, p_5$, and we want to create the policy for the pair $(\{p_2, p_4\}, a_1)$. The corresponding OADD is shown in Figure 1. Notice the ordering of inner nodes from the root to the leaf follows the fixed ordering.

For every pair in the condition-action list we associate an OBDD to the corresponding simple policy by converting actions at the leaves to True. From this perspective, Algorithm 2 computes the overall policy, using the following notation:

- **OBDD**(pol): Convert the OADD pol to an OBDD.
- **policy**(ψ, a): Return the pair's OADD policy.
- **ITE**$(obdd, pol_1, pol_2)$: Return the OADD policy corresponding to the ITE operation.
- $L.pop()$: Pop and return the last element of L.

Theorem 3. The structured policy constructed by Algorithm 2 satisfies the opportunistic property.

Proof sketch. Consider the case where there exists a state s such that $s \models \mathcal{R}^*[G, \vec{a_1}]$ and $s \models \mathcal{R}^*[G, \vec{a_2}]$, where $\vec{a_1}$ (resp. $\vec{a_2}$) is a valid linear suffix with a_1 (resp. a_2) as the first action (and the one chosen in the construction of the condition-action list). Assume that $\vec{a_2}$ is *shorter* than $\vec{a_1}$, and no shorter valid linear suffix exists for the state s.

Since Algorithm 1 adds a (ψ, a) pair to L for every unique condition of a suffix at a particular size, both $(\mathcal{R}^*[G, \vec{a_1}], a_1)$ and $(\mathcal{R}^*[G, \vec{a_2}], a_2)$ will appear in L. Since the size of

$\vec{a_2}$ is shorter than $\vec{a_1}$, $(\mathcal{R}^*[G, \vec{a_2}], a_2)$ must appear before $(\mathcal{R}^*[G, \vec{a_1}], a_1)$ in L. The semantics of ITE used on line 4 allows us to conclude that if $s \models \mathcal{R}^*[G, \vec{a_2}]$, then a_2 would be returned, not a_1. □

Without loss of generality, the proof assumes $a_2 \neq a_1$. Note that an action may appear in multiple pairs in L.

4 Evaluation

We evaluate the claim that employing a POP and monitoring it using our POP Method can provide enhanced flexibility at execution time compared to the EM of a sequential plan using a standard EM method. To do so, we provide both an analytical and experimental analysis of our approach compared to a standard approach for monitoring sequential plans; the Sequential Method (cf. Section 2). We use five domains from the International Planning Competition (IPC) to illustrate the advantage of using our approach: Depots, Driverlog, TPP, Rovers, and Zenotravel. We also investigate the relevant features of a POP through three expository domains: Parallel, Dependent, and Tail.

Experiments were conducted on a Linux desktop with a two-core 3.0GHz processor. Each run was limited to 30 minutes and 1GB of memory. Plans for the Sequential Method were generated by FF [Hoffmann and Nebel, 2001], and a corresponding POP for the POP Method was generated by relaxing unnecessary ordering constraints in the sequential plan to produce a so-called deordering [Bäckström, 1998]. We found that the deordering algorithm we used (originally due to [Kambhampati and Kedar, 1994]) tended to produce the minimum deordering of the plan. While this approach may generate a POP that is fundamentally different from those generated by a traditional POP algorithm, we found that deordering is generally far more practical than computing the POP from scratch.

4.1 Policy Efficiency

To measure impact that using a policy can have for EM, we consider a POP that represents only one linearization. In such a case, the POP Method and Sequential Method will return the same action for any given state. Since we can use *any* valid POP for Algorithm 1, we can feed in the sequential plan, and then pass the resulting condition-action list to Algorithm 2 for the construction of the *Sequential Policy.* Since the Sequential Policy is able to query all conditions in the sequential plan with a single traversal of an OADD, the time required to return an action should be faster than the Sequential Method.

We refer to the *ratio of effort* as the total time for the Sequential Method to return a result for every state in a predefined set of 500 states, divided by the total time for the Sequential Policy to return the same actions. Figure 2 gives an indication of the time savings of our approach. Sorted based on the ratio of effort, the x-axis includes every problem from the five IPC domains. The y-axis indicates the ratio of effort for a given problem. For each problem, the same 500 random states were used for both approaches. The Sequential Policy is 2 to 17 times faster, and the gains become more pronounced with larger plans. With a mean ratio of 6, the use of a structured policy can have substantial gains when it comes to reacting quickly. While the absolute gains are small (on the order of milliseconds at times), the relative speedup may prove to be crucial for real-time applications such as RoboCup. In such domains, the agent must evaluate the state and decide on an action several times a second.

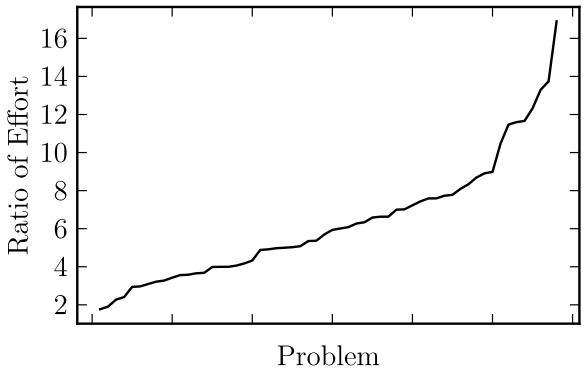


Figure 2: Efficiency of querying the structured policy. The y-axis indicates the total time for the Sequential Method (on 500 random states) divided by the total time for the Sequential Policy. The x-axis ranges over all five IPC problem sets, and is sorted based on the y-axis value.

4.2 Analytical Results

In Section 1, we argued that a POP provides flexibility and robustness at run time. In this analysis we try to quantify the added flexibility afforded by the POP in concert with the POP Method, relative to the Sequential Method. We refer to the number of complete states for which an approach is capable of returning an action as the *state coverage*. We can measure the state coverage by using model counting procedures on the constructed OADD. In the case of the Sequential Method, we generate the OADD as in the previous section. The number of models for either OADD corresponds to the number of states for which the approach can return an action. Figure 3 shows the relative state coverage for the five IPC domains (the POP Method coverage divided by the Sequential Policy coverage).

The y-axis indicates the ratio of states covered for a given problem: state coverage of the POP Method divided by the state coverage of the Sequential Method. For example, a value of 1.5 indicates that the POP Method returns an action in 50% more states than the Sequential Method. We sort problems from each domain based on their y-axis value. The relative state coverage (or *coverage ratio*) ranges from 1 (i.e., the same number of states are handled) to 2.5. Larger plans do not necessarily have a higher coverage ratio, and we conjecture that the ratio has more to do with the structure of a POP, than its size. The state coverage is an approximation since the set of states used in the model count include states that will never occur in practice, either because they are inconsistent or unreachable. Nonetheless, the coverage ratio gives us an approximate measure of the relative gain the flexibility of a POP has to offer when realized with our proposed approach.

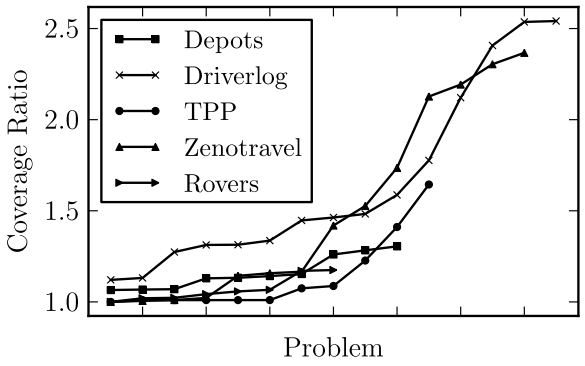


Figure 3: Analytic comparison of state coverage for the POP Method and the Sequential Method. The y-axis indicates the state coverage of the POP Method divided by the state coverage of the Sequential Method. We sort problems from each domain based on their y-axis value.

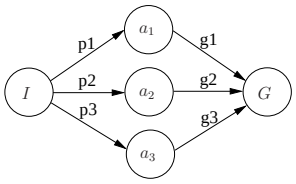


Figure 5: Example of the Parallel domain with $k = 3$.

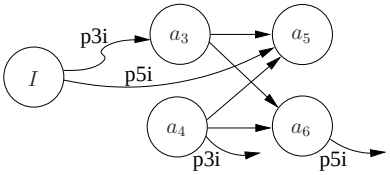


Figure 6: Excerpt from the POP of a Dependent domain problem. An edge with no endpoint signifies that the action has that fact as an effect. We only label edges of interest.

4.3 Expository Domains

The evaluation above was performed on IPC domains which were designed to be challenging domains for sequential planning algorithms. As such, there tends to be significant dependencies between actions and the number of action orderings is large. The high level of dependency is not present in a variety of real-world planning applications (e.g., distributed plans for multiple agents). To evaluate our EM approach for POP, we designed three expository domains that emphasize features we expect to find in real-world planning problems.

Parallelism

The *Parallel* domain demonstrates the impact of multiple linearizations. We construct the Parallel domain so that a solution has k actions that can be performed in parallel. Each action has a single precondition satisfied by the initial state and a single add effect required by the goal. There are no ordering constraints among the actions, and an example with $k = 3$ is shown in Figure 5.

As a consequence of the having no ordering constraints, a solution to a problem in the Parallel domain has a large number of linearizations; with k parallel actions, there are $k!$ linearizations. If the actions mostly have different preconditions and effects, the POP Method will be applicable in many more states than the Sequential Method. There are many states that the Sequential Method fails to capture because of the limited number of unique conditions present in any single linearization. Every linear solution to a Parallel problem has this property. In contrast, the POP Method captures the condition for every linearization suffix. Consequently, we find an exponentially increasing gap in state coverage.

The coverage ratio was computed for problems in the Parallel domain with k ranging from 2 to 10. We present the results in Figure 4a. A clear exponential trend in the increase of state coverage occurs as we increase k.

Extra Support

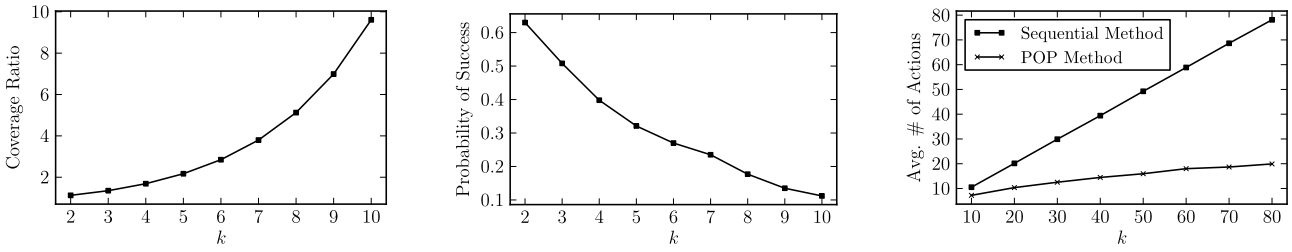
An action has *extra support* if, for a precondition p, there are multiple actions in the POP that act as the achiever of p in at least one linearization. A POP is said to have extra support if one of its actions has extra support. We construct problems in the *Dependent* domain to require k successive pairs of actions such that one action ($a_{\dashv}$) has an extra precondition satisfied both by the initial state and the other action in the pair ($a_{\vdash}$).

An excerpt from a solution to a problem in the Dependent domain is shown in Figure 6. The initial state satisfies the precondition $p3i$ of action a_3. The precondition can also be satisfied by a_4 in any linearization that has a_4 ordered before a_3. If the dynamics of the world cause $p3i$ to become False during execution prior to executing a_3, then we must execute a_4 in order for a_3 to be executable.

The achiever of a fact needed for extra support will depend on the linearization. Monitoring multiple linearizations, the POP Method is a more robust EM solution. Here we measure robustness by how likely an approach is to achieve the goal in a dynamic world. At every time step, we set a randomly selected fact to False or do nothing if it is already False. We then query the approach and execute the action returned. The simulation repeats these two steps, and ends when either the current state satisfies the goal or the approach cannot return an action. We measure the likelihood of reaching the goal as the percentage of trials that end in the goal state.

For a problem in the Dependent domain with a given k, there are 2^k linearizations of the POP. Only one of these will have a 100% success rate when using the Sequential Method: the linearization that correctly orders every pair of actions so that $a_{\vdash}$ comes before $a_{\dashv}$. Using the default linearization generated by FF (which orders $a_{\vdash}$ *after* $a_{\dashv}$), we ran 1000 trials for both approaches. Figure 4b shows the result.

Informally, we can see that the likelihood of reaching the goal approaches zero for the Sequential Method. As the plans become longer, there is more opportunity for something to go wrong due to the dynamics of the world. Since there is always at least one linearization that will get us to the goal, the POP Method always succeeds.



(a) Parallelism (b) Extra Support (c) Critical Orderings

Figure 4: Expository domain results. The x-axis indicates the parameter k used to construct the problem. (a) Coverage ratio of the POP Method and the Sequential Method in the Parallel domain. (b) Likelihood of the Sequential Method to reach the goal in the Dependent domain. (c) Mean number of actions needed by each approach to reach the goal in the Tail domain.

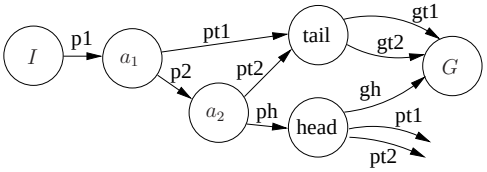


Figure 7: Example of the Tail domain with $k = 2$. An edge with no endpoint indicates an unused action effect.

Critical Orderings

A *critical ordering* is any pair of unordered actions in a POP that must be ordered in a particular way for the Sequential Method to work well. If two linearizations differ only in the ordering of the critical pair of actions, then the Sequential Method for one linearization will outperform the Sequential Method for the other. Since the POP Method simultaneously handles all linearizations, the ordering is irrelevant. We construct the *Tail* domain such that k sequential actions each provide a single precondition to the "tail" action. However, there is also a "head" action that follows the k initial actions and can produce all of the required preconditions for tail. The only two actions left unordered are the head and tail actions. An example with $k = 2$ is shown in Figure 7.

Unlike the Dependent domain, it may be beneficial to have a given ordering even when the agent will always reach the goal. We investigate the performance of the two approaches in a simulation where at every time step we set a randomly selected fact to True or do nothing if it is already True. The POPs have the property that any linearization will reach the goal eventually when using the Sequential Method. What is of interest is how *quickly* the goal is achieved.

When using the Sequential Method for the linearization that has head ordered after tail, positive fact flips have little impact on the number of steps to reach the goal. The other linearization has the advantage of being able to *serendipitously jump* into a future part of the plan. We show the mean number of steps for each approach on eight instances in Figure 4c.

We see a clear trend for the Sequential Method that suggests it requires roughly k actions to reach the goal. In contrast, the number of actions required on average for the POP Method grows very slowly – the final problem taking only a quarter of the actions to reach the goal on average. The POP Method is able to skip large portions of the plan when a fact changes from False to True. The Sequential Method, on the other hand, must continue executing almost the entire plan.

4.4 Discussion

The state coverage for the IPC domains was not as great as the coverage for the expository domains because they impose significant constraints on action orderings and thus do not highlight the flexibility afforded by POPs and exploited in our EM approach. In general it is beneficial to use our approach, even for a single linearization. However, we have identified two scenarios in which our approach fails. First, in the Parallel domain with k actions, there are $2^k - 1$ unique conditions. While this number is far less than the $k * k!$ possible suffixes, it is large enough to become unwieldy for $k > 15$. Second, even if the POP is a single linearization, interactions between the ordering of the add effects and preconditions, along with the fact ordering in the construction of the OADD, can result in an exponential blow up of the policy size. We have not observed this behavior in our experiments. Both scenarios suggest that in a richer domain, it would be interesting to investigate a trade-off between the size of the policy representation and its robustness.

5 Concluding Remarks

In this paper we examined the problem of monitoring the execution of a POP. Due to its structure, a POP compactly represents a family of plans that share actions but allows for numerous (at the extreme, an exponential number of) different linearizations. Our objective was to develop a means of POP EM that would seamlessly switch between these different plans based on the current state of the world, and thus maximally exploit the flexibility of the POP. EM of sequential plans typically attempts to determine whether a plan remains valid with respect to the state of the world. We defined the objective of POP EM as determining POP *viability* and characterized the conditions under which a POP was viable by relating them to goal regression over all linearization suffixes of the POP. Acknowledging the inefficiency of such a computation, we developed a more efficient algorithm that employed goal regression but exploited shared POP substructure to do so efficiently. We proved the correctness of this algorithm with respect to POP viability. Then, rather than

employ these "relevant conditions" as input to a typical EM algorithm, we employed these conditions in the construction of a structured policy – an OADD that takes a state as input and returns an action to perform in reaction to the state of the world. In so doing, the policy combines two phases of EM – state evaluation and action selection – into one system.

We evaluated our POP EM system both analytically and experimentally with the objective of assessing the claim that employing a POP, rather than a sequential plan, could provide notably enhanced flexibility at execution time. Experiments were run on IPC domains and on expository domains. On the IPC domains (which were designed to be more constrained than many real-world applications and as such are less able to exploit the least commitment property of POPs) our approach is able to continue executing in up to 2.5 times the number of states as the sequential-plan-based approach. The speed-up in identifying an action of our POP Policy compared to the sequential-plan-based approach is up to a factor of 17. Our expository domains highlight various properties that affect POP EM. In these domains, we demonstrated an exponential increase in the number of states where the POP remains viable relative to the sequential counterpart.

There are commonalities between our approach and work on the topic of dynamic controllability (e.g., [Shah *et al.*, 2007]): our shared methodology is to compile plans into a dispatchable form for online execution. However, whereas the dynamic controllability work generally mitigates for uncertainty in the execution time for actions, our work addresses unexpected change in the environment. Our work complements dynamic controllability by focusing on a different source of uncertainty. In many real world scenarios, both sources of uncertainty appear and we intend to explore synergies between these two approaches.

Also related to our work are a number of systems for approximate probabilistic planning. For example, ReTrASE [Kolobov *et al.*, 2009] uses regression in a similar fashion to build a "basis function" that provides the condition for which a plan succeeds in a deterministic version of a probabilistic planning problem. This use of determinization followed by regression is related to previous work [Sanner and Boutilier, 2006; Gretton and Thiébaux, 2004] which uses first-order regression on optimal plans over small problems to construct a policy for larger problems in the same domain.

There are several extensions to our approach which we hope to pursue. The computational effort required for constructing the OADD is typically the bottleneck, and using a more concise representation of the policy may provide significant gains. When the POP Method is unable to return an action, we could instead attempt to re-plan using the information already embedded in the policy. Doing so opens the door to a variety of re-planning strategies, and suggests there may be merit in finding a more intelligent construction of the OADD. There is the potential to develop a better informed heuristic for computing a sequential plan that will produce a POP with the properties that lead to good performance in our EM approach. We also plan to investigate the use of our POP Method with probabilistic planners such as FF-Replan, where we would hope to see a reduction in the number of re-plans required during execution.

Acknowledgements

The authors gratefully acknowledge funding from the Ontario Ministry of Innovation and the Natural Sciences and Engineering Research Council of Canada (NSERC). We would like to thank Christian Fritz, and the anonymous referees for useful feedback on earlier drafts of the paper.

References

[Bäckström, 1998] C. Bäckström. Computational aspects of reordering plans. *Journal of Artificial Intelligence Research*, 9(1):99–137, 1998.

[Bahar *et al.*, 1997] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Formal methods in system design*, 10(2):171–206, 1997.

[Boutilier *et al.*, 1995] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1104–1113, 1995.

[Fikes *et al.*, 1972] R.E. Fikes, P.E. Hart, and N.J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

[Fritz and McIlraith, 2007] C. Fritz and S. McIlraith. Monitoring plan optimality during execution. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, pages 144–151, 2007.

[Gretton and Thiébaux, 2004] C. Gretton and S. Thiébaux. Exploiting first-order regression in inductive policy selection. In *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence (UAI-04)*, pages 217–225, 2004.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14(1):253–302, 2001.

[Kambhampati and Kedar, 1994] S. Kambhampati and S. Kedar. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence*, 67(1):29–70, 1994.

[Kolobov *et al.*, 2009] A Kolobov, Mausam, and D Weld. ReTrASE: Integrating paradigms for approximate probabilistic planning. *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1746–1753, 2009.

[Sanner and Boutilier, 2006] S. Sanner and C. Boutilier. Practical linear value-approximation techniques for first-order MDPs. In *Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence (UAI-06)*, 2006.

[Shah *et al.*, 2007] J. Shah, J. Stedl, B. Williams, and P. Robertson. A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, pages 296–303, 2007.

[Veloso *et al.*, 1998] M.M. Veloso, M.E. Pollack, and M.T. Cox. Rationale-based monitoring for planning in dynamic environments. In *Proceedings of the Fourth International Conference on AI Planning Systems (AIPS-98)*, pages 171–179, 1998.

[Waldinger, 1977] R. Waldinger. Achieving Several Goals Simultaneously. *Machine Intelligence*, 8:94–136, 1977.

[Weld, 1994] D.S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27, 1994.

[Wilkins, 1985] D.E. Wilkins. Recovering from execution errors in SIPE. *Computational Intelligence*, 1(1):33–45, 1985.