# SODA POP : Robust online Partial Ordering Planning for real life applications

Antoine GRÉA Samir AKNINE Laetitia MATIGNON

*Abstract*— **This is the best of all abstracts ever !**

**It consists of two paragraphs but that is more than enough for it to be awesome.**

## INTRODUCTION

For some time Partial Order Planning (POP) has been the most popular approach to planning resolution. The way the search is made using flexible partial plan as a search space allowed for more versatility for a wide variety of uses. As of more recent years, new state space search models and heuristics [1] have been demonstrated to be more efficient than POP planners due to the simplicity and relevance of states representation opposed to partial plans [2]. This have made the search of performance the main axis of research for planning in the community.

While this goal is justified, it shadows other problems that some practical applications causes. For example, for applications needing online planning the flexibility of Plan Space Planning (PSP) is an obvious advantage : plans can be repaired on the fly as new informations and objectives enter the system . Some other cognitive applications may provide approximate plans to be refined. These can contain errors and imperfections that will decrease significantly the efficiency of the computation and the quality of the output plan. Others AI applications needs complete plans that details the reasons behind a failure and what are its consequences on the rest of the steps to achieve the provided goal.

These problems call for new ways to improve the answer and behavior of a planner in order to provide relevant plan information pointing out exactly what needs to be done in order to make a planning problem solvable, even in the case of obviously broken input data. We aim to solve this in the present paper while preserving the advantages of PSP algorithms. Our Soft Ordering and Defect Aware Partial Ordering Planning (SODA POP) algorithm will target those issues.

## I. PLAN SPACE PLANNING DEFINITIONS

In order to present our work and explain examples we will introduce a way of representation for schema and notations for mathematical representation.
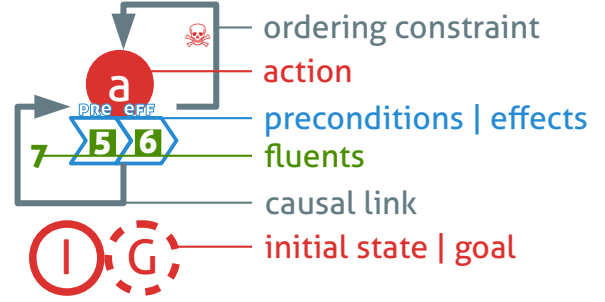


Fig. 1. Global legend for how partial plans are represented in the current paper

### A. Fluents

A fluent is a property of the world. It is often represented by first order logical propositions but in our case we choose to focus on the algorithm and to represent fluents as simple literals (fully instantiated). We note $\neg f$ the complementary fluent of $f$ meaning that $f$ is true when $\neg f$ is false and vice-versa.

In order to make our example simpler we use $\mathbb{Q}^*$, the set of relative integers without 0, as the fluent domain. We use negative integers to represent opposite fluents.

### B. State

We define a state as a set of fluents. States can be additively combined. We note $s_1 + s_2 = (s_1 \cup s_2) - \{f | f \in s_1 \wedge \neg f \in s_2\}$ such operation. It is the union of the fluents with an erasure of the complementary ones.

### C. Action

An action is a state operator. It is represented as a tuple $a = \langle pre, eff \rangle$ with $pre$ and $eff$ being sets of fluents, respectively the preconditions and the effects of $a$. An action can be used only in a state that verifies its preconditions. We note $s \models a \Leftrightarrow pre(a) \subset s$ the verification of an action $a$ by a state $s$.

An action $a$ can be functionally applied to a state $s$ following :

$$ a := \begin{smallmatrix} \{s \models a | s \in S\} \to S \\ a(s) \mapsto s + eff(a) \end{smallmatrix} $$

with $S$ being the set of all states. There are some special names for actions. An action with no preconditions is synonymous to a state and one with empty effect is called a goal.

## D. Partial Plan

A *partial plan* is a tuple $p = \langle A_p, L \rangle$ where $A_p$ is a set of steps (actions) and $L$ is a set of causal links of the form $a_i \xrightarrow{f} a_j$, such that $\{a_i, a_j\} \subset A_p \wedge f \in eff(a_i) \cap pre(a_j)$ or said otherwise that $f$ is provided by $a_i$ to $a_j$ via this causal link. We include the ordering constraints of PSP in the causal links. An ordering constraint is noted $a_i \rightarrow a_j$ and means that the plan consider $a_i$ as a step that is prior to $a_j$ without specific cause (usually because of threat resolution).

## E. Problem

We define a partial plan satisfaction problem as a tuple noted $P = \langle A, I, G, p \rangle$ with :

- $I$ and $G$ being the pseudo actions representing respectively the initial state and the goal.
- $p$ being a partial plan to refine.
- $A$ the set of all actions.

## F. Flaws

When refining a partial plan, we need to fix flaws. Those could be present or be created by the refining process. Flaws can either be unsatisfied subgoals or threats to causal links.

*1) Subgoals:* A subgoal $s$ is a precondition of an action $a_s \in A_p$ with $s \in pre(a_s)$ that isn't satisfied by any causal link. We can note a subgoal as :

$$a_i \xrightarrow{s} a_s \notin L \mid \{a_i, a_s\} \subset A_p$$

A resolver for a subgoal is an action $a_r \in A$ that has $s$ as an effect $s \in eff(a_r)$. It is inserted along with a causal link noted $a_r \xrightarrow{s} a_s$.

*2) Threats:* A step $a_t$ is said to threaten a causal link $a_i \xrightarrow{t} a_j$ if and only if

$$\neg t \in eff(a_t) \wedge a_i \succ a_t \succ a_j \models L$$

Said otherwise, the action has a possible complementary effect that can be inserted between two actions needing this fluent while being consistant with the ordering constraint in $L$.

The usual resolvers are either $a_t \rightarrow a_i$ or $a_j \rightarrow a_t$ which are called respectively promotion and demotion links. Another resolver is a called a white knight that is an action $a_k$ that reestablish $t$ after $a_t$

## G. Consistency

A partial plan is consistent if it contains no ordering cycles. That means that the directed graph formed by step as vertices and causal links as edges isn't cyclical. This is important to guarantee the soundness of the algorithm.

## H. Flat Plan

We can instantiate one or several flat plans from a partial plan. A flat plan is an ordered sequence of actions $\pi = [a_1, a_2 \ldots a_n]$ that acts like a pseudo action $\pi = \langle pre_\pi, eff_\pi \rangle$ and can be applied to a state $s$ using functional composition operation $\pi := \bigcirc_{i=1}^{n} a_n$.

We call a flat plan valid if and only if it can be functionally applied on an empty state. We note that this is different from classic state planning because in our case the initial state is the first action that is already included in the plan.

## I. Solution

A partial plan which generates only valid flat plans is a solution to the given problem.

## II. CLASSICAL POP

Partial Order Planning (POP) is a popular implementation of the general PSP algorithm. It is proven to be sound and complete [3]. The completeness of the algorithm guarantees that if the problem has a solution it will be found by the algorithm and the soundness assures that all answer from the algorithm is valid. POP refines a partial plans by trying to fix its flaws.

## A. Description

From that point the base algorithm is very similar for any implementation of POP : Using an agenda of flaws that is efficiently updated after each refinement of the plan. A flaw is selected for resolution and we use a non deterministic choice operator to pick a resolver for the flaw. The resolver in inserted in the plan and we recursively call the algorithm on the new plan. On failure we return to the last non deterministic choice to pick another resolver. The algorithm ends when the agenda is empty or when there is no more resolver to pick for a given flaw.

## B. Limitations

This standard way of doing have seen multiple improvement over expressiveness like with UCPOP [4], hierarchical task network to add more user control over sub-plans [5], cognition with defeasible reasoning [6], or speed with multiple ways to implement the popular fast forward method from state planning [7]. However, all these variants doesn't treat the problem of online planning, resilience and soft solving. Indeed, these problems can affect POP's performance and quality as they can interfere with POP's inner working.

Before, continuing we present a simple example of classical POP execution with the problem represented in figure
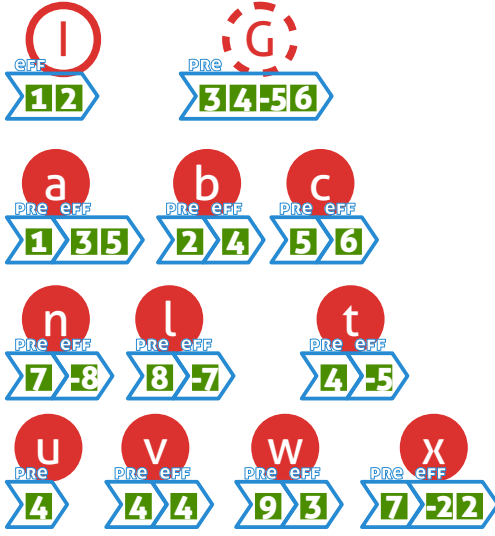
Fig. 2. A simple problem that will be used throughout this paper

2. We didn't represented empty preconditions or effects to improve readability. Here we have an initial state $I = \langle \emptyset, \{1, 2\} \rangle$ and a goal $G = \langle \{3, 4, -5, 6\}, \emptyset \rangle$ encoded as dummy steps. We also introduce actions that aren't steps yet but that are provided by $A$. The action $a$, $b$ and $c$ are normal actions that are useful to achieve the goal. The action $t$ is meant to be threatening to the plan's integrity and will generate threats. The actions $u$, $v$, $w$ and $x$ are toxic actions. We introduce $u$ and $v$ as useless actions, $w$ as a dead-end action and $x$ as a contradictory action.
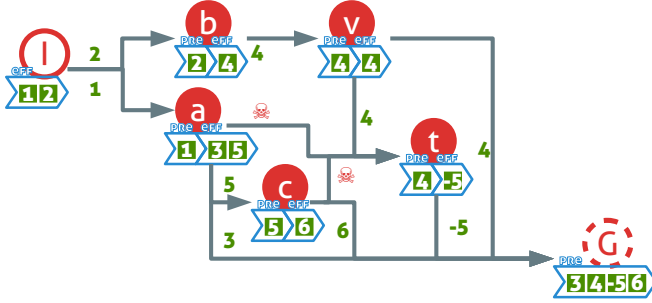


Fig. 3. Standard POP result to the problem

As explained, this problem has been crafted to illustrate problems with standard POP implementations. We give the resulting plan of standard POP in figure 3. We can see some problems as for how the plan has been built. The action $v$ is being used even if it is useless since $b$ already provided fluent 4. We can also notice that despite being contradictory the action $x$ raised no alarm. As for ordering constraints we can clearly see that the link $a \rightarrow t$ is redundant with the path $a \xrightarrow{5} c \rightarrow t$ that already put that constraint by transitivity. Also some problems arise durring execution with the selection of $w$ that causes a dead-end.

All these issues are caused by what we call defects as they aren't regular PSP flaws but still causes problems with

execution and results. We will address defects and how to fix them in the next section.

## III. Auxiliary Algorithms to POP

In order to improve POP algorithms' resilience, online performance and plan quality we propose a set of auxiliary algorithms that provides POP with a clean and efficiently populated initial plan.

### A. Proper plan generation

As in online planning goals can be known in advance, we add a new mechanism that generates proper plans for goals. We define for that the concept of participating action. An action $a$ participates in a goal $G$ if and only if $a$ has an effect $f$ that is needed to accomplish $G$ or that is needed to accomplish another participating action's preconditions. A proper plan is a partial plan that contains all participating actions as steps and causal links that bind them with the step they are participating in. This proper plan is independent form an initial step that we might not have at the time of that generation.

This auxiliary algorithm is used as a caching mechanism for online planning. The algorithm starts to populate the proper plan with a quick and incomplete backward chaining.

This can independently be replaced with a quick forward chaining in other applications and thus benefit from the advantage of state planning and the latest improvement over fast-forward.

The aim here is to efficiently populate most of the plan without guarantee about completeness and soundness. This way we can make POP much more efficient as most of the selection is done by the time it starts.

### B. Defect resolution

When the POP algorithm is used to refine a given plan (that wasn't generated with POP or that was altered), a set of new defects can be present in it interfering in the resolution and sometimes making it impossible to solve. To clarify, those aren't regular POP flaws but new problems that classical POP can't solve. The aim of this auxiliary algorithm is to clean the plans from such defects in order to improve computational time, resilience and plan quality. It should be noted that in some cases cleaning plans will increase the number of flaws in the plan but will always improve the overall quality of it.

There are two kinds of defects : The illegal defects that violates base hypothesis of PSP and the interference defects that can lead to excessive computational time and to poor plan quality.

*1) Illegal defects:* These defects are usually hypothesized out by regular models. They are illegal use of partial plan representation and shouldn't happen under regular circumstances. They may appear if the input is generated by an approximate cognitive model that doesn't ensure consistency of the output or by unchecked corrupted data.

*a) Cycles:*

A plan cannot contain cycles as it makes it impossible to complete. Cycles are usually detected as they are inserted in a plan but poor input can potentially contains them and break the POP algorithm as it cannot undo cycles.

We use a popular and simple Depth First Search (DFS) algorithm to detect cycles. Upon cycle detection the algorithm can remove arbitrarily a link in the cycle to break it. The most effective solution is to remove the link that is the farthest from the goal travelling backward as it would be that link that would have been last added in the regular POP algorrithm.

*b) Inconsistent actions:*

In a plan some actions can be illegal for POP. Those are the actions that are contradictory. An action $a$ is contradictory if and only if

$$\{f, \neg f\} \in eff(a) \vee \{f, \neg f\} \in pre(a)$$

We remove only one of those effect or preconditions based on the usage of the action in the rest of the plan. If none of those are used we choose to remove both.

*c) Toxic actions:*

These actions are those that have effects that are already in their preconditions. This can damage a plan as well as make the execution of POP algorithm much longer than necessary. They are defined as :

$$a|pre(a) \cap eff(a) \neq \emptyset$$

This is fixed of one of two ways : if the action has only some of its fluent toxic ($pre(a) \nsubseteq eff(a)$) then the toxic fluents are removed following $eff(a) = eff(a) - pre(a)$, otherwise the action is removed alltogether from plan and $A$.

*d) Liar links:*

The defects can be related to incorrect links. The first of which are liar links : a link that doesn't reflect the preconditions or effect of its source and target. We can note

$$a_i \xrightarrow{f} a_j | f \notin eff(a_i) \cap pre(a_j)$$

These can form with the way inconsistent actions are fixed : a deleted fluent could still have links in the plan.

To resolve the problem we either replace $f$ with a saviour : a fluent in $eff(a_i) \cap pre(a_j)$ that isn't already provided or we delete the link all together.

*2) Interference defects:* These class of defects are not as toxic as the illegal ones : they won't make the plan unsolvable but they can still cause performance drops in POP execution. These can appear more often in regular POP results as they are not targeted by standard implementations.

*a) Redundant links:*

This defect can happen in POP generated plans to some extends. A redundant link have a transitive equivalent of longer length. That means that a link $a_i \to a_j$ is redundant if and only if it exists another path from $a_i$ to $a_j$ of length greater to 1. Since POP relies on those additional links, this part focus on removing the ones that were generated for threat removal purpose to simplify the plan.

*b) Competing causal links:*

Causal links can be found to compete with one another. This can't happen in POP but won't be fixed by the algorithm. A competing link $a_i \xrightarrow{f} a_k$ competes with another link $a_j \xrightarrow{f} a_k$ if it provides the same fluent to the same action.

We remove the link that has the lower outgoing degree of their source. If they have the same outgoing degree we choose on the higher incoming degree. This ensure that the least useful action get cut from the plan as a useless action.

*c) Useless actions:*

Actions can sometimes have no use in a plan as they don't contribute to it. It is the case of orphans actions (actions without any links) and in a completed plan actions that are dead ends (action with no outgoing path to the goal). We also consider useless actions that doesn't have effects (except the goal step).

*C. Soft Resolution*

This auxiliary algorithm is meant to deal with failure. It will heal the plan to make the failure recoverable for the next iteration of POP. Of course it can't fix the plan by keeping the problem as it is. This of course breaks some properties as the algorithm no longer adhere to the specification of the input, but in exchange it will always issue a valid plan whatever happens. For more information on this property go take a look at the appropriate section bellow.

Soft failure is useful when the precision and validity of the output isn't the most important criteria we look for. In some cases (like in recognition processes) it is more useful to have an output even if it is not exact than no output at all. That is why we propose a soft failing mechanism for POP algorithms.

We need to define a few new notions first then we will explain the algorithm.

*1) Needer:* A needer is an action that is needing resolution related to a flaw. The needer of a subgoal is the action that has an unsatisfied precondition and for a threat it is the target of the threatened causal link.

*2) Proper fluents:* A proper fluent of a flaw is the flew that caused the flaw. For a subgoal it is the unsatisfied precondition and for a threat it is the fluent of the threatened causal link.

*3) Healers:* The concept of healer is made to target rogue flaws that caused total failure. A healer is a forged resolver that solve the need for a given flaw. The general formula of a healer is the following :

$$a_h \xrightarrow{f} a_n$$

with $a_n$ the needer of the flaw and the crafted action $a_f = \langle \emptyset, f \rangle$ with $f$ being the proper fluent of the flaw.

For threats we need an additional healer specified as $a_t \to a_h$ with $a_t$ being the threatening action.

*4) Violation degree:*

*5) Healing process:* The method is to keep track of reversions in the algorithm by storing the partial plan and the unsatisfiable flaw each time a non deterministic choice fails. We note the set of these failed plans $F$. As the POP algorithm encounter a final failure this auxiliary algorithm get invoked. The aim is to evaluate each backtracking partial plan to choose the best one.

Once the algorithm fails completely the soft failing algorithm kicks in and choose the best plan. To sort incomplete plans we use the length of the plan and the degree of violation.

The length of a plan is the number of actions in the plan's graph : $|P| = |A_P|$ and the violation degree is the number of flaws remaining divided by the number of links in the plan. The quality of a plan is the length of it times the violation degree.

The better plan is then selected to be refined by previous defect resolution method. It is then inputed again in POP algorithm and we recursively repeat the steps. This stops when the plan doesn't change between two iterations or if we reached a defined number of iterations. The resulting plan is returned with violation information about it for the caller to handle.

## IV. Properties of the algorithms

After defining the way our algorithms works we will focus on the properties that can be achieve by combining them together.

### A. Convergence of POP

As to our knowledge no proof of the convergence of POP has been done we want to explicitly formulate one.

The classic planning problem is already proven to be decidable without functions in the fluents [2]. Therefore we can categorise the termination cases. In the case of a solvable problem, POP is proven to be complete. This ensures convergence in that case. Now for the more complex case of unsolvable problems we need to refer to the way POP works. POP algorithm will seek to solve flaws. At any time there is a finite number of flaws since the plans have a finite number of steps. As POP resolves these flaws it will either continue until resolution or until failure. The problem is that POP can encounter loops in the dependancies of actions or in threats resolution. These loops can't occur in POP algorithm since a cycle in the ordering constraints instantly causes a failure as the plan isn't consistent anymore. This prooves that POP always converges.

### B. Hyper soundness

Now that we proved that regular POP converges we can introduce the next property : hyper soundness. An algorithm is said to be hyper sound when it gives a valid solution for all problems including unsolvable ones. We note that this property isn't compatible with consistency regarding the original problem but still does regarding the derived problem that includes all fake actions in $A$.

The hyper soundness of our combined algorithm is proven using the convergence of POP and the way the Soft solving behaves. As a POP fails it will issue flawed partial plans. As we fix the flaws artificially we make sure that this failure won't happen again in the next iteration of POP on the fixed plan. As the number of flaws is finite and POP converges, the whole algorithm will converge with all flaws solved therefore issuing a valid plan.

### C. Enhancement for online planning

## V. Experimental results

## Conclusion

## References

[1] S. Richter, M. Westphal, and M. Helmert, "LAMA 2008 and 2011," in *International Planning Competition*, 2011, pp. 117–124.

[2] M. Ghallab, D. S. Nau, and P. Traverso, *Automated planning: theory and practice*. Amsterdam ; Boston: Elsevier/Morgan Kaufmann, 2004.

[3] K. Erol, J. A. Hendler, and D. S. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning." in *AIPS*, 1994, vol. 94, pp. 249–254.

[4] J. S. Penberthy, D. S. Weld, and others, "UCPOP: A Sound, Complete, Partial Order Planner for ADL." *Kr*, vol. 92, pp. 103–114, 1992.

[5] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal, "HiPOP: Hierarchical Partial-Order Planning," in *STAIRS 2014: Proceedings of the 7th European Starting AI Researcher Symposium*, 2014, vol. 264, p. 51.

[6] D. R. García, A. J. García, and G. R. Simari, "Defeasible reasoning and partial order planning," in *Foundations of Information and Knowledge Systems*, Springer, 2008, pp. 311–328.

[7] A. J. Coles, A. Coles, M. Fox, and D. Long, "Forward-Chaining Partial-Order Planning." in *ICAPS*, 2010, pp. 42–49.