SODA POP : Robust 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.

Introduction

The Soft Ordering and Defect Aware Partial Ordering Planning algorithm (SODA POP)

I. Plan Space Planning Definitions

To start we need to introduce definitions and notations in order to explain our partial order planning algorithms.

A. Fluents

A fluent is a property of the world. It is often represented by first order logical propositions. In our case we choose to focus on the algorithm and to represent fluents as instantiated propositions (no variables in fluents). We note $\neg f$ the complementary fluent of f meaning that f is true when $\neg f$ is false and vice-versa.

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, one with empty effect is called a goal and the action that have both empty is the empty action noted $a_\emptyset = \langle \emptyset, \emptyset \rangle$.

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 $a_i \to^f a_j$, 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 an effect of the step a_i and a precondition of the step a_j. We include the ordering constraints of PSP in the causal links. An ordering constraint is noted $a_i \to a_j$ 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 :

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

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

2) Threats: A step a_t is said to threaten a causal link $a_i \to^t a_j$ if and only if $\neg t \in eff(a_t)$ and $a_i \succ a_t \succ a_j$ is a possible order given the ordering constraint in L.

The usual resolvers are either $a_t \to a_i$ or $a_j \to a_t$ 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

H. Flat Plan

We can instantiate one or several flat plans from a partial plan. A flat plan is an ordered sequence of actions $\pi = [a_1, a_2 \dots a_n]$ 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 the empty action. We note that this is different from classic planning because in our case the initial state is the first action that is already included in the plan.

I. Solution

A problem is solved once the algorithm finds a partial plan which generate only valid flat plans.

II. Classic PSP Algorithms

A. POP

Partial Order Planning (POP) is a popular implementation of the general PSP algorithm. It is proven to be sound and complete and some implementations even adds expressivity while preserving this property [1]. 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.

The procedure as described in [2] is using an agenda of subgoals to efficiently process them without having to search them at each iteration. It also treats threats differently as they form from inserted subgoals. The steps are rather simple, the algorithm first tries to satisfy a subgoal taken from the agenda. It does so by listing its resolvers and non deterministically choose one. It tries to insert it in the plan and search for new threats created with the new resolver. This is also solved by a list of resolvers for the threats that are also non deterministically chosen. Once done it will call itself on the new partial plan and agenda. If a non deterministic choice fails, it must try the other resolvers until there are no more available to them propagate the failure to the last non deterministic choice.

We need to point out that the choice of scanning only threats generated by subgoals breaks the property of soundness and completeness of the algorithm when working with non empty plans. The most obvious example is a given plan that already has a threat. The algorithm will compute it without fixing it and will return a non valid plan (soundness) even if there where a correct solution (completeness).

III. Robustness extensions to POP

POP is a complete algorithm but some simpler linear planning was found to be more efficient early on due to research on efficient heuristics. The main reason is the way POP works on plan based search states that makes it hard to guide the search. To account for this problem POP has a significant advantage on linear planning that is its flexibility. Another advantage is that POP is far more efficient at replanning since fixing plan is its way of working. In the domain of real-time planning it is therefore much more adapted to use POP as it will be able to adapt the plan easily on the go.

However, in real life applications the data isn't necessarily "clean" especially when coming from other perception based systems. This is why a wide variety of new problems can arise making the plan unsolvable by traditional means. This calls for ways to find these new *defects* and to solve them.

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 unresolvable defects can be present in it making it impossible to solve. To clarify, those aren't regular POP flaws but new problems that POP can't solve. The aim of this extension is to improve POP's resilience to such mis-constructed plans. There are two kinds of defects : The illegal defects that violates base hypothesis of PSP and the useless defects that only cause the plan to be

A. Illegal defects

1) 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, or said otherwise $a_i \rightarrow^f a_j$ with $f \notin eff(a_i) \cap pre(a_j)$. To resolve the problem we either replace f with a fluent in $eff(a_i) \cap pre(a_j)$ that isn't already provided or we delete the link all together.

2) Cycles: A plan cannot contain cycles as it makes it impossible to complete. Upon cycle detection the algorithm can remove arbitrarily a link in the cycle to break it.

3) Inconsistent actions: In a plan some actions can be illegal for POP. Those are the actions that are contradictory. We note $a|f, \neg f \in eff(a) \vee f, \neg f \in pre(a)$ such actions.

B. Useless defects

1) 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 $a_i \rightarrow a_j$ is redundant if and only if it exists another path from a_i to a_j of length greater or equal 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.

2) 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 $a_i \rightarrow^f a_k$ competes with another link $a_j \rightarrow^f a_k$ if it provides the same fluent to the same action.

3) 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 initial step of a plan).

IV. Soft resolution of unsolvable problems

Here we propose another way to deal with failure. The new algorithm is meant to find relevant solution to unsolvable problems. This of course breaks the property of soundness of POP but is proven to be useful in time sensitive context that needs real time hints on the actions to accomplish or that needs detailed and relevant reasons of failure.

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.

The method is to keep track of reversions in the algorithm by storing the partial plan and the reason it fails each time a non deterministic choice fails. We note the set of these failed plans F. 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.

V. Results

Conclusion

VI. References

[1] 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.

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