SODA POP : 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. Planification Definitions

Let's first define a few notions. A fluent is a property of the world. It can be positive or negative. In our case we ignore the standard way of adding variables to fluents and decide to focus on the algorithm (extensions of the present work might add this later on). We call state 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 by the right side of the contradictory ones. An action is a state operator. It is 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 $a := \frac{\{s \models a | s \in S\} \rightarrow S}{a(s) \mapsto s + eff(a)}$ with S being the set of all states. There are some special names for actions. An action with an empty precondition 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 $a_\emptyset = \langle \emptyset, \emptyset \rangle$.

A partial plan satisfaction problem is a tuple noted $\Pi = \langle I, P, G, A \rangle$ with :

- I and G being the pseudo actions representing respectively the initial state and the goal.
- $P = \langle A_P, L \rangle$ being a partial plan constituted of steps (A_P as a set of actions) and fluent labeled causal links (L). We use the causal link also as ordering constraints.
- A the set of all actions.

We can instantiate one or several flat plans from a partial plan. A flat plan is an ordered sequence of actions $\mathbb{P} = [a_1, a_2 \dots a_n]$ that acts like a pseudo action $\mathbb{P} = \langle pre_{\mathbb{P}}, eff_{\mathbb{P}} \rangle$ and can be applied to a state s using functional composition operation $\mathbb{P} := \bigcirc_{i=1}^{n} a_n$. We call a flat plan valid if and only if it can be functionally applied on the empty action. This is different that classic planning because in our case the initial state is the first action that is already included in the plan.

II. Classic POP Algorithm

Partial Order Planning (POP) is a type of planning algorithm that is sound and complete [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 uses partial plans and tries to fix its flaws. A flaw is either a subgoal that isn't satisfied or a threat to a causal link.

A subgoal s is a fluent in a precondition of an action $a \in A_P$ with $s \in pre(a)$ that isn't satisfied by any causal link. In classic POP the search start with the non deterministic choice of an action $a_s \in A$ that verifies $s \in eff(a_s)$ also noted $a_s \models^s a$. If there isn't such an action then the algorithm fails and tries to return to a previous non deterministic choice. If there are one or more actions that matches, the algorithm add a causal link $a_s \rightarrow^s a$ to L. In the general algorithm variable grounding happens at this step.

A step a_t is said to threatens a causal link $a_i \rightarrow^f a_j$ if and only if $\neg f \in eff(a_t) \wedge \forall \mathbb{P} \in P^{\downarrow}, [a_i, a_t, a_j] \in \mathbb{P}$ with $P^{\downarrow}$ being the set of flattened plans from the current partial plan P. The usual solutions are either demoting or promoting the threat to cancel it. For that the algorithm add an ordering constraint that respectively force the threatening step before or after the link. The promotion is the operation that add the ordering constraint $a_t \rightarrow a_i$ and the demotion adds the ordering constraint $a_j \rightarrow a_t$. A consistency check is needed to confirm the change. The test includes graph cyclicality check : cycles makes the plan inconsistent. It also includes a test for redundancy, checking if the new ordering constraint wasn't already present in the graph. The last check is that we can't promote before the initial step and can't demote after the goal step.

The algorithm will recursively solve all the flaws, if a non deterministic choice fails it goes back to the last to try something else. If all defaults are fixed the plan is valid and it stops the search. If not it means that there are no valid plans and that the problem is impossible.

III. Enhancement of 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. That is one of the reasons that left POP behind research wise. Here we present some novel ways to tweak POP in order to make it faster to compute and also prone to handle further extensions.

A. Single flaw selection

In clasical POP the algorithm make a list of all the flaws and then select them to fix them. In the present version, in order to improve the performances, the algorithm select the first flaw found and fixes it. This allows to end the loops as soon as we found one flaw and also to search for new flaws as the algorithm goes.

B. Dynamic consistency checks and loop detection

One big issue with POP is that some problems can have cycles in their solutions because of cyclical dependencies. These causes the algorithm to loop uncontrollably and to fail for lack of ressources. This is also a reason why POP algorithms aren't convergent. The solution is in dynamic consistency checks. These will make sure that no loops are created and will react if we try to insert a causal link that is already present. These insertions are symptomatic of a loop and therefore triggers a new quarantine mechanism that helps the algorithm to avoid these toxic actions when computing. The result is that the new POP algorithm is sound, complete and convergent.

IV. Extensions to POP

The goal of the present paper is to present extensions to classic POP algorithms.

When POP algorithms are used as replanning techniques, the POP algorithm can fail due to the fact that some defects can be present in the input partial plan. Some defects are illegal for POP (which is why they weren't treated in the original algorithm) such as incoherent actions or liar links. Some are less dangerous but can slow POP or make the plan more complex than it should be. Those are useless actions, competing causal links and redundant links.

All these extensions leads to be able to make this last one : the soft solving of failed plans. This soft behaviour, while inconsistent with the natural soundness of POP, can be proven very useful in applications where precision isn't the main concern and where comprehensive (while incorrect) plans of action are better than nothing. These soft plans are accompanied with a violation degree that gives a numeric cost of the constraints violated.

A. 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 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 misconstructed plans.

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) Redundant links: This defect can happens in POP generated plans to some extends. A redundant links 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.

4) 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.

5) Illegal actions: In a plan some actions can be illegal in 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. We can notice that action containing 0 as a fluent are illegal (this will be of use latter).

6) 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).

B. Soft Plan Generation

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

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.