ABORT: Abstract Building Operating in Real Time as a refinement strategy for hierarchical partial order planning (Draft)

Antoine Gréa

Introduction

à expliquer

pas de ref à POP et HTN dans intro?

The domain of automated planning is considered a basis of artificial intelligence. As such it must adapt to suit very different needs. Some works might require a fast real time scheduling process to build a path in complex environments. Other works will require flexibility for cooperation or competition with other agents. That is the reason behind the diversity of the planning comunity approach wise.

Our research aims at a totally separated domain : intent recognition and robotics. In the recent years several works extended what is known in psycology as the *theory of mind*. That theory suposes that to recognise other's intents and goals we often use to transpose our own. That is like saying "*what would **I** do ?*" when observing actions of another person. This leads to new ways to use *invert planning* as an inference tool.

One of the first to propose that idea was Baker et al. [1] that uses bayesian planning to infer intentions. Another one was Ramirez et al. [2] that found an elegant way to transform a plan recognition problem into classical planning. This is done simply by encoding observation constraints into the planning domain [3] to ensure the selection of actions in the order that they were observed. A cost comparison will then give us a probability of the goal being pursued given the observations. Some works extended this with multi-goal recognition [4] and robotic applications [5].

Very recently, another paper extended this approach significantly. The method proposed by Sohrabi et al. [6] make the recognition at the fluent level instead of actions. It assign costs to missing or noisy observed fluents by using soft constraints (often called user preferences). This method also uses a meta-goal that combine each possible goal and is realised when at least one of them is reached. *paragraphe pas clair*

Sohrabi et al. states that the quality of the recognition is directly linked to the quality of the generated plans. This is why guided diverse planning was prefered along with the posibilities to infer several possible goals at once.

Since our applicative domain also include robotics, we wish to account also for the real time aspect of the prediction. Indeed, a prediction is only useful *before* it happens. The human aspect of our context also asks for high level description and extensive domains.

Obviously these work hypothesis are in direct contradiction : the harder the problem, the longer the resolution. And this is especially true in automated planning as it has been proven to be a P-SPACE problem if not harder. A big and complex domain is intractable on limited embeded robotics. *ref*

Our interest was on the following question : what happens when one *can't* find the solution to the problem ? What if the problem can't be solved ? What if we ran out of time ? This question has been partially threated in [7] where "excuses" are being investigated as response to unsolvability.

TODO : Introducing our work *détailler les excuses si pas fait dans la suite; préciser différence avec ton travail*

etant donne la suite, il faut expliquer HTN et POP car cela apparait ensuite dans les définitions

Figure 1: A potential logo. (I love logos)

? pas clair

1 Definitions

In order to explain how our method works, the basics of notation and definitions used in the rest of the paper needs to be laid.

Table 1: Most used symbols in the paper. **FIXME: Update when needed**

Symbol	Description
$pre(o), eff(o)$	Preconditions and effects of the operator o.
$\mathcal{D}, \mathcal{P}$	Planning domain and problem.
$methods(o)$	Methods of the operator o.
$lv(o), lv(\mathcal{D})$	Level of the operator or domain.
$\phi^{\pm}(l)$	Signed incidence function for partial order plans.
source step of l	ϕ^{-} gives the source and ϕ^{+} the target.
	No sign gives a pair corresponding to link l.
$causes(l)$	Gives the causes of causal link l.
$L^{\pm}(s)$	Set of incoming (L^{-}) and outgoing (L^{+}) links of step s.
	No sign gives all adjacent links.
$l \downarrow s$	Link l participates in the partial support of step s.
$\ddagger_f s$	Fluent f isn't supported in step s.
$\pi \Downarrow s$	Plan π fully supports s.
	If no left side it means just full support.

ajouter > contrainte d'ordre entre steps d'un plan

1.1 Notations

In this paper, we use the notation defined in table 1. Our notations are adapted from the ones used in [8] and [7]. We use the symbol $\pm$ to signify that the notation is signed. All related notions will be defined later.

1.2 Domain

Planners often work in two times : first we input the planning domain then we give the planner an instance of a planning problem to solve. For this work we used a custom compiler that is inspired by PDDL and RDF notations.

Definition 1 (Domain)**.** A domain $\mathcal{D} = \langle C_{\mathcal{D}}, R, F, O \rangle$ is a tuple where :

- $C_{\mathcal{D}}$ is the set of **domain constants**.
- R is the set of **relations** (also called properties) of the domain. These relations are similar to quantified predicates in classical works.
- F is the set of **fluents** used in the domain to describe operators.
- O is the set of **operators**.

We note $lv(\mathcal{D})$ the abstraction level of the domain specified as $max_{o \in O}(lv(o))$.

Example: The domain we will use as example is a variant of the classical kitchen domain. This domain is more detailed and contains composite operators.

as-tu une ref pour ce domaine ? Dire en une phrase ou 2 en quoi consiste ce domaine, ou donner un extrait du domaine

A domain is mostly made by operators. All other components are byproducts of their declaration. In our case, we need composite operators in order to be able to do HTN planning.

HTN devra être évoqué en intro …

Definition 2 (Operator)**.** An operator o is an instanciable action of the domain. It can have instanciation parameters. We note it $o = \langle name, pre, eff, methods, lv \rangle$ where :

- $name$ is the **name** of the operator.
- pre and eff are sets of fluents that are respectively the **preconditions and effects** of the operator.
- $methods$ is a set of **methods** that can realize the operator. A method is a partial order plan that has initial and goal steps of the form $init(o) = \langle$"$init$", $pre(o), pre(o), \emptyset, 0\rangle$ and respectively with $goal$ and eff. This formulation allows execution of planners on composite actions and eases their expansion in existing plans.
- lv is the **abstraction level** of the operator.

partial order plan et step défini plus tard !

Erreur dans init ?

manque def de method ?

An operator with no method is said *atomic* and will have an abstraction level of 0. In all other case the operator is *composite* and its abstraction level is defined as $lv(o) = \max(lv(s)) + 1$ with $s \in S_m$ being a step of the method $m \in methods(o)$.

step of method ?

Example: In the kitchen the action $take(thing)$ has a precondition of the parameter not being taken and an effect of the parameter being taken. As it has no method, it is atomic and of level 0.

exemple d'un operator composite ?

All the planning depends heavily on fluents and the unification between states (sets of fluents).

states: à expliquer/définir en une phrase

FIXME it may be possible to do the paper without defining fluents.

Definition 3 (Fluent)**.** A fluent is a parameterized relation $f = r(arg_1, arg_2, ..., arg_n)$ where :

- r is the relation of the fluent under the form of a boolean predicate.
- $arg_i | i \in [1, n]$ are the arguments (possibly quantified).
- $n = |r|$ is the arity of r.

Fluents are signed. Negative fluents are noted $\neg f$ and behave as a logical complement. Quantifiers are affected by the sign of the fluents. We use an open world hypothesis with fluents being not satisfied until provided whatever their sign.

FIXME check the definition of "open world hypothesis" not being assuming all positive fluents.

Example: The precondition of $take(thing)$ is just a single negative fluent noted $\neg taken(thing)$.

1.3 Problem

The other part of the input of most planners is the problem instance. This problem is often most simply described by two components : the initial state and the goal of the problem.

Definition 4 (Problem)**.** The planning problem is defined as a tuple $\mathcal{P} = \langle \mathcal{D}, C_{\mathcal{P}}, \Omega \rangle$ where :

- $\mathcal{D}$ is a planning domain.
- $C_{\mathcal{P}}$ is the set of **problem constants** disjoint from the domain constants.
- Ω is the problem's **root operator** which methods are solutions of the problem. *?*

The root operator contains the initial state and goal (noted respectively I and G) of the problem. *préciser que cela est contenu car i et G sont les effets et precond de l'opérateur.*

Example: We use a simple problem for our example domain. The initial state provides that nothing is ready, taken or boiled and all containers are empty (all using quantifiers). The goal is to have tea ready and with sugar. For reference, here is the problem as stated in our file :

```
1     init eff (ready(~), taken(~),
      boiled(~), * ~(in) *);
2     goal pre (ready(tea), sugar in cup);
```

pas plutôt le présenter en langage classique (PDDL) pour ne pas perdre le lecteur ?

In all cases, the root operator is initialized at $\Omega = \langle "", s_0, s^*, \{m_{lv(\mathcal{D})}\}, lv(\mathcal{D}) + 1 \rangle$ with s_0 being the initial state and s^* the goal specifications.

pourquoi ne pas réutiliser I et G au lieu de s0 et s ?*

The method $m_{lv(\mathcal{D})}$ is a partial order plan with only the initial and goal steps linked together. Partial order plans are at the heart of Partial Order Planning (POP). *ok à évoquer en intro*

Definition 5 (Plan)**.** A partially ordered plan is a directed graph $\pi = (S, L)$ with : *S_\pi et L_\pi ?*

- S the set of **steps** of the plan as vertices. A step is either an operator or an instanciated action. Partial instanciation is allowed.
- L the set of **causal links** of the plan as edges.

We note $l : a_s \xrightarrow{c} a_t$ the link between its source a_s and target a_t caused by c. In our representation causal links are also ordering links. If we need a causeless ordering link we simply omit the cause.

Example: The initial partial order plan is $m_{lv(\mathcal{D})} = (\{I, G\}, \{I \rightarrow G\})$ with :

- $I = \langle "init", \emptyset, s_0, \emptyset, 0 \rangle$ and
- $G = \langle "goal", \emptyset, s^*, \emptyset, 0 \rangle$.

2 The ABORT planner

Our approach is based upon POP and, more precisely, a HTN version of POP [9]. When extending POP for hierarchical resolution, it is best to simply use the system of flaws by introducing a new flaw called abstraction flaw. The driving force of POP is a good flaw and resolver selection. The order and timing is the most important factor a quick resolution.

TODO read about commitment

2.1 Classical POP

First of all, here is a small refresher about the classical POP algorithm. Partial Order Planning fixes flaws in a partial plan to refine it into a valid plan that is a solution to the given problem.

Once all flaws solved it is stated that a plan is valid. This was proven in [10] by extending on the definition of *full support* of all conditions in a plan.

Definition 6 (Support)**.** An open condition f of a step s is supported in a partial order plan π if and only if $\exists l \in L_\pi^-(s) \wedge \nexists s_b \in S_\pi : f \in causes(l) \wedge (\phi^-(l) \succ s_b \succ s \wedge \neg f \in eff(s_b))$. This support is noted $\pi \downarrow_f s$. **Full support** of a step is achieved when all its preconditions are supported $\pi \Downarrow s \equiv \forall f \in pre(s) : \pi \downarrow_f s$. *en plus de la définition formelle, expliquer ce que cela signifie !*

The classical flaws are exclusively dealing with support related problems.

Definition 7 (Flaws)**.** A flaw in a partial plan is a contradiction with its validity. Flaws have a *proper fluent* f and a step often called the *needer* s_n. Those are the problematic entities that prevent the resolution of the problem. There exist two types of classical flaws :

- **Subgoals** are a lack of support for a needer of the subgoal. We can note them $\ddagger_f s_n$.
- **Threats** are caused by steps that can break a causal link with their effects. They are called *breakers* of the threat-

Remarque pour futur article:
est-ce que la notion de support est quelque chose de connu dans la litterature de planning ? Je trouve cela beaucoup moins compréhensible que les définitions « classiques » de menaces et sous-buts … as-tu un intérêt à utiliser/introduire ici cette notion ?
Ici la menace se comprend bien, mais le sous-buts qui ré-utiise ta notion de support n'est pas très claire, car la notion de support ne l'est pas.
SI la notion de support ne te sert pas après c'est à enlever, car ca rajoute des notions complexes pour rien. Garder juste les 2 types de flaws classiques (pourquoi avec une notation pour chq si nécessaire)
Si elle est importante pour la suite, il faut au minimum expliquer dans des termes « simples » ce que cela signifie.

De même pour les notations concernant le support: si tu les réutilises ensuite, il faut les garder, sinon, cela ne sert à rien d'introduire ces notations supplémentaires (pour l'instant ces notations n'apparaissent pas dans la suite c'est pour cela que je fais cette remarque).

ened link. A step s_b is threatenning a causal link $s_p \xrightarrow{f} s_n$ if and only if $s_b \neq s_p \neq s_n \wedge \neg f \in eff(s_b) \wedge s_p \succ s_b \succ s_n$.

Example: In our initial plan, there are two unsupported subgoals : one to make the tea ready and another to put sugar in it. In this case the needer is the goal step and the proper fluents are each of its preconditions.

These flaws need fixing before the plan can become a solution. In POP it is done by finding resolvers.

Definition 8 (Resolvers)**.** Classical resolvers are additional causal links. It is a kind of mirror image of flaws.

- For subgoals, the resolvers are a set of potential causal links containing the proper fluent f in their causes while having the needer step s_n as their target.
- For threats, their are usually only two resolvers : *demotion* and *promotion* of the breaker relative to the threatened link.

Example: the subgoal for sugar in our example can be solved by using the action $pour(sugar, cup)$ as the source of a causal link carying the proper fluent as its only cause.

A plan cannot be solution until all of the flaws are fixed. Sometimes, that cannot be achieved because of constraints in the plan. Sometimes it is because the side effects of the resolver application causes incompatible flaws.

Definition 9 (Side effects)**.** Flaws that arise because of the application of a resolver are called *related flaws*. They are inserted in the flaws agenda whith each application of a resolver :

non défini avant

- *Related subgoals* are all the new open conditions inserted by new steps.
- *Related threats* are the causal links threatened by the insertion of a new step or deletion of a guarding link.

Flaws can also become irrelevant when a resolver is applied. It is always the case of the targeted flaw but can also affect other flaws. Those *invalidated flaws* are removed from the agenda upon detection :

- *Invalidated subgoals* are subgoals satisfied by the new causal links or removal of needer.
- *Invalidated threats* happen when the breaker no longer threaten the causal link because order got guaranteed or the causal link or breaker have been removed.

Example: adding the action $pour(sugar, cup)$ causes another subgoal with each of the action's preconditions which are that the cup and the sugar must be taken and sugar not already in the cup.

In that last definition we introduced elements that aren't present in classical POP, namely *negative resolvers*. All classical resolvers only add elements to the partial plan. This is because, in our method, we introduce a new flaw and resolver that can be negative. *? quand parles-tu de cela avant ?*

2.2 Abstraction in POP

As stated earlier, we use a variant of POP with abstract actions. This variant is strongly inspired from the works of Bechon *et al.* on a planner called HiPOP [9]. This work explains adaptation of POP on HTN domains. In order to do that we must add a new abstraction flaw.

Definition 10 (Abstraction flaw)**.** An abstraction flaw is when the partial plan conatins a step with a non zero level. This step is the needer of the flaw.

- *Resolvers* : An abstraction flaw is solved with an **expansion resolver**. The resolver will replace the needer with one of its instanciated methods in the plan. This is done by linking all causal links to the initial and goal step of the method as such : $L^-(init(m)) = L^-(s_n) \wedge L^+(goal(m)) = L^+(s_n)$ with $m \in methods(s_n)$. This means that all incoming links will be redirected to the initial step of the method and all outgoing links will be linked from the goal of the method. **TODO schema of the resolution**
- *Side effects* : An abstraction flaw can be related to the introduction of a composite action in the plan by any resolvers and invalidated by its removal.

Example: When adding the step $make(tea)$ in the plan to solve the subgoal that needs tea being made, we also introduce an abstraction flaw that will need the step replaced by its method using an expansion resolver.

One of the main focus of HiPOP is to handle the issues caused by hierarchical domains when solved with POP. These issues are mostly linked to the way the expansion resolver might introduce new flaws and the optimal order in which solving these issues.

One way this is solved is by always chosing to solve the abstraction flaw in priority. While this may arguably make the

full resolution faster it also lose oportunities to obtain abstract plans in the process.

2.3 Abstract plans

In our work we chose to delay the resolution of the abstraction flaws after all other flaws have been solved. This gives our planner the ability to compute abstract plans that satisfy the classical definition of validity in POP. Indeed these plans are devoid of classical flaws and therefore classically valid.

Example: When planning in the kitchen domain, HiPOP will chose the action $make(tea)$ which is composite and expand it in the next refinement. This is akin to a search in depth. Our method will solve all the other flaws first and then change of abstraction level while keeping the abstract plan. This plan is akin to $init \rightarrow make(tea) \rightarrow pour(sugar, cup) \rightarrow goal$.

However, this plan isn't a completely valid plan. This might come as a surprise but that kind of partial plan are very useful in a variety of ways. They are semantically richer, especially for human. Mathematically they form an excelent approximation of the final plan, especially since their methods are instanciated with the composite action and are available for analysis. Last but not least : these plans are obtained in a fraction of the time needed to obtain the full plan and allows for anytime partial solving of the problem with increasing precision.

Example: In the case of intent recognition using inverted planning, the method counts the number of fluents that should and shouldn't be observed in order to compute the probability. In the example plan, the level of $make(tea)$ is 1. Meaning that with this method we have actually 100% accuracy on the first level for intent recognition (theres no additional actions to add at level 0). All this in 1% of the time needed to solve the complete plan.

FIXME: far from the final formulation ofc...

TODO: way more schema for all that example

3 Properties of ABORT

4 Results

Conclusions

References

[1] C. L. Baker, J. B. Tenenbaum, and R. R. Saxe, "Goal inference as inverse planning," in *CogSci*, 2007.

[2] M. Ramirez and H. Geffner, "Plan recognition as planning," in *IJCAI*, 2009, pp. 1778–1783.

[3] M. Baioletti, S. Marcugini, and A. Milani, "Encoding planning constraints into partial order planning domains," in *KR*, 1998, pp. 608–616.

[4] J. Chen, Y. Chen, Y. Xu, R. Huang, and Z. Chen, "A Planning Approach to the Recognition of Multiple Goals," *IJIS*, vol. 28, no. 3, pp. 203–216, 2013.

[5] K. Talamadupula, G. Briggs, T. Chakraborti, M. Scheutz, and S. Kambhampati, "Coordination in human-robot teams using mental modeling and plan recognition," in *IRoS*, 2014, pp. 2957–2962.

[6] S. Sohrabi, A. V. Riabov, and O. Udrea, "Plan Recognition as Planning Revisited," in *IJCAI*, 2016.

[7] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel, "Coming Up With Good Excuses: What to do When no Plan Can be Found." in *ICAPS*, 2010, pp. 81–88.

[8] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[9] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal, "HiPOP: Hierarchical Partial-Order Planning," in *STAIRS*, 2014, vol. 264, p. 51.

[10] 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.