HEART: HiErarchical Abstraction for Real-Time partial order planning

Paper #1788

Abstract

When asked about the biggest issues with automated planning, the answer often consists of speed, quality, and expressivity. However, certain applications need a mix of these criteria. Most existing solutions focus on one area while not addressing the others. We aim to present a new method to compromise on these aspects in respect to demanding domains like assistive robotics, intent recognition and real-time computations. The HEART planner is an anytime planner based on a hierarchical version of Partial Order Planning (POP). Its principle is to retain a partial plan at each abstraction level in the hierarchy of actions. While the intermediary plans are not solutions, they meet criteria for usages ranging from explanation to goal inference. This paper also evaluates the variations of speed/quality combinations of the abstract plan in relation to the complete planning process.

Introduction

In the context of our research on intent recognition with assistive robots, we faced one of the most widespread issue of artificial intelligence : how to be relevant within time constraints ? Indeed, our software must be able to execute on robots with limited processing power and to give answers in real-time (within a polling frequency). All the while it needs to interact with humans in the most natural way possible requiring the software to provide simplified answers.

Why an automated planner ? Planners aren't meant to solve recognition problems. However, several works extended what is known in psychology as the *theory of mind.* That theory supposes that to recognize the intents and goals of others we often simply transpose our own. That is like saying "*what would **I** do I were them ?*" when observing the behavior of another person. This leads to new ways to use *inverted planning* as an inference tool. One of the first to propose that idea was Baker et al. [Baker, Tenenbaum, & Saxe, 2007] that uses Bayesian planning to infer intentions. Another one was Ramirez et al. [Ramirez & Geffner, 2009] 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 [Baioletti, Marcugini, & Milani, 1998] 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 to be pursued made the observations. Some works extended this with multi-goal recognition [J. Chen, Chen, Xu, Huang, & Chen, 2013] and even robotic applications [Talamadupula, Briggs, Chakraborti, Scheutz, & Kambhampati, 2014]. A new method, proposed by Sohrabi et al. [Sohrabi, Riabov, & Udrea, 2016], makes the recognition at the fluent level instead of actions. It assigns costs to missing or noisy observed fluents by using soft constraints (often called user preferences). This method also uses a meta-goal that combines each possible goal and is realized when at least one of them is reached. Sohrabi et al. state that the quality of the recognition is directly linked to the quality of the generated plans. This is why guided diverse planning was preferred along with the possibilities to infer several possible goals at once.

What happens when the planner runs out of time ? Obviously, extensive problems needs more computation time in order for a solution to be found. this is especially true in automated planning as it has been proven to be a P-SPACE problem if not harder[Weld, 1994]. Big problems are usually intractable within the tight constraints of real-time robotics. The problem of dealing with unsolvability have already been addressed in [Göbelbecker, Keller, Eyerich, Brenner, & Nebel, 2010] where "excuses" are being investigated as potential explanations for when a problem have no solutions. The closest way to address time unsolvability is by using explainability [Fox, Long, & Magazzeni, 2017]. In our context we do not need the complete solution as much as an explication of the plan.

What kind of answers do we really need ? For intent recognition an important metric is the number of correct fluents. So if finding a complete solution is impossible, partial solution can meet enough criteria to give good aproximation of the goal probability. In classical planning, one of the main approach is called Partial Order Planning (POP). It works by refinining partial plans into a solution. Another approach is Hierarchical Task Networks (HTN) that are meant to process the problem using composite actions in order to define hierarchical tasks within the plan. These two approaches are not exclusive and have been combined in several works [Pellier & Fiorino, 2007, Gerevini, Kuter, Nau, Saetti, & Waisbrot [2008]]. Our work is based on HiPOP [Bechon, Barbier, Infantes, Lesire, & Vidal, 2014], that is expanding the classical POP algorithm with new flaws in order to make it solve HTN problems. This kind of planner allows for flexible plans with high abstraction actions that which fits our requirements.

In the rest of the paper we will describe how HiErarchical

Table 1: Most of the symbols used in the paper.

Symbol	Description
$pre(a), eff(a)$	Preconditions and effects of the action a.
$methods(a)$	Methods of the action a.
$\mathcal{D}, \mathcal{P}$	Planning domain and problem.
$lv(a), lv(\mathcal{D})$	Abstraction level of the action or domain.
$width(\mathcal{D})$	Width of a domain.
$\phi^{\pm}(l)$	Signed incidence function for partial order plans. ϕ^- gives the source and ϕ^+ the target step of l. No sign gives a pair corresponding to link l.
$causes(l)$	Gives the causes of a causal link l.
$a_a \succ a_s$	Step a_a is anterior to the successor step a_p.
$L^{\pm}(a)$	Set of incoming (L^-) and outgoing (L^+) links of step a. No sign gives all adjacent links.
$l \downarrow a$	Link l participates in the partial support of step a.
$\ddagger_f\ a$	Fluent f isn't supported in step a.
$\pi \Downarrow a$	Plan π fully supports a. If no left side it means just full support.
A^n_x	Proper actions set of x down n levels. A_x for $n = 1$ and A^*_x for $n = lv(x)$.
$[exp]$	Iverson brackets : 0 if $exp = false$, 1 otherwise.

Abstraction for Real-Time partial order planning (HEART) can create abstract intermediary plans that can be used for intent recognition within our time constraints.

1 Definitions

In this paper, we use the notations defined in table 1. Our notations are adapted from the ones used in [Ghallab, Nau, & Traverso, 2004], [Göbelbecker et al., 2010] and graph theory. The symbol $\pm$ is only used to signify that the notation have signed variants. All related notions will be defined later.

Planners often work in two phases : first we input the planning domain then we give the planner an instance of a planning problem to solve.

1.1 Domain

The domain is the context of the planner. It specifies the allowed operators that can be used to plan and all the fluents they use for preconditions and effects.

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 first order logic.
- F is the set of **fluents** used in the domain to describe operators.
- O is the set of **operators** which are fully lifted *actions*.

Example: The domain in listing 1 we use as an example is inspired from the kitchen domain of [Ramirez & Geffner, 2010] modified to suit our needs.

```
take(item) pre (taken(~), ?(item));
take(item) eff (taken(item));
heat(thing) pre (~(hot(thing)),
    taken(thing));
heat(thing) eff (hot(thing));
pour(stuff,into) pre (stuff ~(in) into,
    taken(stuff));
pour(stuff,into) eff (stuff in into);
put(utensil) pre (~(placed(utensil)),
    taken(utensil));
put(utensil) eff (placed(utensil),
    ~(taken(utensil)));
infuse(extract,liquid,container) ::
    Action;
make(drink) :: Action;
```

Listing 1: Domain file used in our planner. Composite actions have been truncated for lack of space.

A domain contains the definitions of all fluents that can be used in the operators.

Definition 2 (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. The quantifiers are affected by the sign of the fluents. We don't use the closed world hypothesis : fluents are not satisfied until provided whatever their sign.

Example: When we want to describe an item not being held we can use the fluent $\neg taken(item)$. If the cup contains water $in(water, cup)$ is true.

The central notion of planning is operators. Instanciated operators are called *actions*. In our framework, actions can be partially instantiated and therefore we consider operators as a special case of actions.

Definition 3 (Action)**.** An action a is an instantiated operator of the domain. It can have instantiation parameters. We note it $a = \langle name, pre, eff, methods\rangle$ where :

- $name$ is the **name** of the action.
- pre and eff are sets of fluents that are respectively the **preconditions and effects** of the action.
- $methods$ is a set of **methods** (partial order plans) that can realize the action.

Example: The precondition of the operator $take(item)$ is simply a single negative fluent noted $\neg taken(thing)$ ensuring the variable $thing$ isn't already taken.

Composite actions are represented using methods. Each method is a partial order plan used in Partial Order Planning (POP). An action without methods is called *atomic*.

Definition 4 (Plan)**.** A partially ordered plan is an *acyclic* directed graph $\pi = (S, L)$ with :

- S the set of **steps** of the plan as vertices. A step is an action belonging in the plan.
- 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 *ordering constraints* are defined as the transitive cover of causal links over the set of steps. We note ordering constraints like this : $a_a \succ a_s$ with a_a being *anterior* to its *successor* a_s. Ordering constraints can't form cycles meaning that the steps must be different and that the successor can't also be anterior to its anterior steps : $a_a \neq a_s \wedge a_s \nsucc a_a$.

If we need to enforce order, we simply add a causal link without cause. This graph representation along with the implicit ordering constraints makes for a simplified framework that still retains classical properties needed for POP.

Our representation in files for methods is very simplified and only give causeless causal links. The causes of each links to the preconditions and effects of the composite action are infered. This inference is done by running an instance of POP on the method (keeping the composite action as intended) while compiling the domain. We use the following formula to compute the final result : $pre(a) = \bigcup_{a_s \in L^+(a)} causes(a_s)$ and $eff(a) = \bigcup_{a_s \in L^-(a)} causes(a_s)$. Errors are reported if POP cannot be completed or if one of the two following rules are not met :

- Methods must contain an initial and goal step along with a guarante of the respect of order constraints between the two ($I_m \succ G_m$)
- and they must not contain their parent action as step or as a step of any composite actions within them ($a \notin A^*_a$, see section 3.1).

1.2 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 5 (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.

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 :

```
init eff (hot(~), taken(~), placed(~), ~
    in ~);
goal pre (hot(water), tea in cup, water
    in cup, placed(spoon), placed(cup));
```

Listing 2: One problem instance used as an example for this paper.

In all cases, the root operator is initialized at $\Omega = \langle "", s_0, s^*, \{\pi\} \rangle$ with s_0 being the initial state and s^* the goal specifications.

The method π 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).

Example: The initial partial order plan is $\pi = (\{I, G\}, \{I \to G\})$ with :

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

The goal of the planner is to refine the plan π until it becomes a solution of the problem.

1.3 Partial Order Planning

At the base of our method is the classical POP algorithm. It works by refining a partial plan until no more flaws are present and therefore it becomes a solution of the planning problem.

Definition 6 (Flaws)**.** A flaw in a partial plan is a contradiction with its validity. Flaws have a *proper fluent* f and a related step often called the *needer* a_n. There exist two types of classical flaws :

- **Subgoals** are *open conditions* that are yet to be supported by another step a_n often called *provider*.
- **Threats** are caused by steps that can break a causal link with their effects. They are called *breakers* of the threatened link. A step a_b is threatening a causal link $a_p \xrightarrow{f} a_n$ if and only if $\neg f \in eff(a_b) \wedge a_p \succ a_b \succ a_n$. Said otherwise, the breaker can cancel an effect of a providing step before it gets used by its needer.

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 7 (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 taking the needer step s_n as their target.
- For threats, there 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 $pour(water, cup)$ as the source of a causal link carrying the proper fluent as its only cause.

A plan cannot be a 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 8 (Side effects)**.** Flaws that arise because of the application of a resolver are called *related flaws*. They are inserted in the *agenda* (a set of flaws supporting flaw selection) with each application of a resolver :

- *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 has been removed.

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

In that last definition we mentioned effects that aren't present in classical POP, namely *negative effects*. All classical resolvers only add elements to the partial plan. Our method needs to occasionally remove steps so we plan ahead accordingly.

Our version of the POP algorithm is based on [Ghallab et al., 2004, sec. 5.4.2]. In algorithm 1 we present our generic version of POP.

Algorithm 1 Partial Order Planner

```
function pop(Agenda $a$, Problem $\mathcal{P}$)
    if $a = \emptyset$ then        ▷ Populated agenda of flaws needs to be provided
        return Success        ▷ Stops all recursion
    Flaw $f \leftarrow$ choose($a$)        ▷ Chosen flaw removed from agenda
    Resolvers $R \leftarrow$ solve($f$, $\mathcal{P}$)
    for all $r \in R$ do        ▷ Non deterministic choice operator
        apply($r$, $\pi$)        ▷ Apply resolver to partial plan
        Agenda $a' \leftarrow$ update($a$)
        if pop($a'$, $\mathcal{P}$) = Success then        ▷ Refining recursively
            return Success
        revert($r$, $\pi$)        ▷ Failure, undo resolver application
    $a \leftarrow a \cup \{f\}$        ▷ Flaw wasn't resolved
    return Failure        ▷ Revert to last non deterministic choice of resolver
```

2 Approach

We aim to demonstrate the uses of abstraction in refinement based planners. In order to do this we need to explain a few additional notions regarding abstraction and then explain our algorithm in more details.

2.1 Abstraction in POP

In order to handle abstraction with POP there are a couple of ways. The most straight forward way is illustrated in another planner called Duet [Gerevini et al., 2008] by simply managing hierarchical actions separately from a regular planner. We chose another way strongly inspired by the works of Bechon *et al.* on a planner called HiPOP [Bechon et al., 2014]. This planner is adapting HTN notions for POP by extending it and modifying its behavior. It does this by adding new flaws and resolvers.

Definition 9 (Abstraction flaw)**.** An abstraction flaw is when the partial plan contains 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 instantiated methods in the plan. This is done by linking all causal links to the initial and goal step of the method as such: $L^-(I_m) = L^-(s_n) \wedge L^+(G_m) = L^+(s_n)$ 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.
- *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.

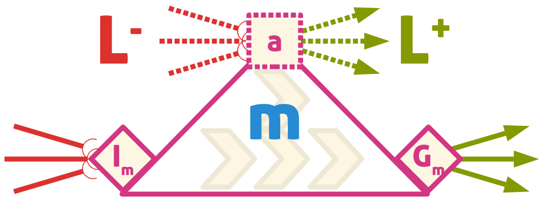

Figure 1: Example of an expansion of a composite action a using one of its methods m. The elements with dashed lines are removed after the expansion.

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 focuses 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 handled is by always choosing to solve the abstraction flaws first. While this may arguably make the full resolution faster, it also loses opportunities to obtain abstract plans in the process.

2.2 Heart beats

In order to simplify further expressions, we define two other notions :

Definition 10 (Proper actions)**.** Proper actions are actions that are "contained" within an entity : * For a *domain* or a *problem* it is $A_{\mathcal{D}|\mathcal{P}} = O$. * For a *plan* it is $A_\pi = S_\pi$. * For an *action* it is $A_a = \bigcup_{m \in methods(a)} S_m$. That can be extended down several levels recursively: $A_a^n = \bigcup_{a' \in A_a} A_{a'}^{n-1}$.

We note $A_a^* = A_a^{lv(a)}$ the complete set of all proper actions of the action a.

Example: The proper actions of $make(drink)$ are the actions contained within its methods. Its complete set of proper actions is that plus all proper actions of its single proper composite action $infuse(drink, water, cup)$.

Definition 11 (Abstraction level)**.** This is a measure of the maximum amount of abstraction an entity can hold :

$$lv(x) = \left(\max_{a \in A_x}(lv(a)) + 1 \right) [A_x \neq \emptyset]$$

(we use Iverson brackets here, see notations in table 1).

Example: The abstraction level of any atomic action is 0 while it is 2 for the composite action $make(drink)$. The example domain has an abstraction level of 3.

The main focus of our work is toward obtaining **abstract plans** which are plans that are completed while still containing abstract actions. We compute them in cycles called heart beats.

Definition 12 (Heart beat)**.** A heart beat is a cycle of planning where all abstraction flaws are delayed. Each cycle have a designated *abstraction level* that limits the resolver selection for subgoals. Only operators of level less than or equal to this level are allowed to be inserted in the plan. Once no more flaws other than abstraction flaws are present in the agenda, the current plan is saved and all remaining abstraction flaws are solved at once. Each heart beat produces a more detailed abstract plan than the one before.

This allows the planner to do an approximative form of anytime execution. At anytime the planner is able to return a fully supported plan. Before the first heart beat, the plan returned is the following $I \xrightarrow{s_0} \Omega \xrightarrow{s^*} G$. We use the root operator to indicate that no heart beats have been completed. These intermediary plans are not solutions of the problem. However, they have some interesting properties that make them useful for several other applications.

Example: In our case using the method of intent recognition explained in [Sohrabi et al., 2016], we can already use this plan to find a likely goal explaining an observation (a set of temporally ordered fluents). That can make an early assessment of the probability of each goal of the recognition problem.

In order to find the solution the HEART planner needs to reach the final heart beat of abstraction level 0.

3 Properties

First we need to prove that our approach conserve the properties of classical POP when given enough time to complete.

3.1 Soundness

For an algorithm to be sound, it needs to provide only *valid* solutions. Our approach can provide invalid plans but that happens only on interruptions and is clearly stated in the returned data. In order to prove soundness we first need to define the notion of support.

Definition 13 (Support)**.** An open condition f of a step a is supported in a partial order plan π if and only if $\exists l \in L^-_\pi(a) \wedge \nexists a_b \in S_\pi : f \in causes(l) \wedge \left(\phi^-(l) \succ a_b \succ a \wedge \neg f \in eff(a_b)\right)$. This means that the fluent is provided by a causal link and isn't threatened by another step. We note this $\pi \downarrow_f a$.

Full support of a step is achieved when all its preconditions are supported : $\pi \Downarrow a \equiv \forall f \in pre(a) : \pi \downarrow_f a$.

We also need to define validity in order to derive all the equivalences of it :

Definition 14 (Validity)**.** A plan π is a valid solution of a problem $\mathcal{P}$ if and only if $\forall a \in S_\pi : \pi \Downarrow a \wedge lv(a) = 0$.

It is reminded that $G \in S_\pi$ by definition and that it is an atomic action.

We can now start to prove the soundness of our approach. We base this proof upon the one done in [Erol, Hendler, & Nau, 1994]. It states that for classical POP, if a plan doesn't contain any flaws it is fully supported. Our main difference being with abstraction flaws we need to prove that its resolution doesn't leave classical flaws unsolved in the resulting plan.

Lemma (Expansion with an empty method)**.** *If a composite action a is replaced by an empty method $m = (\{I_m, G_m\}, \{I_m \to G_m\})$, replace a with I_m in all needers of existing flaws and we add all open conditions of G_m as subgoals the resulting plan will not have any undiscovered flaws.*

Proof. The initial and goal step of a method are *transparent* ($pre(a) = eff(a)$).

$$L^-(I_m) = L^-(a) \wedge pre(I_m) = pre(a) \implies (\pi \Downarrow a \equiv \pi \Downarrow I_m)$$

If we do as stated, all subgoals are populated for I_m and G_m. For the threats, the order constraints are preserved and therefore can't cause another threat (the link between I_m and G_m is causeless). □

Lemma (Expansion with an arbitrary method)**.** *If a composite action a is replaced by an arbitrary method m that contains the order constraint $I_m \succ G_m$ and replace a with I_m in all needers of existing flaws and we add all open conditions contained within S_m as subgoals and all threatened links within L_m as threats, the resulting plan will not have any undiscovered flaws.*

Proof. Adding to the previous proof, if a new method is inserted in an existing plan when replacing an action we actually do the following operation on steps: $S'_\pi = (S_\pi \setminus a) \cup S_m$. In the worst case, we will need to introduce one subgoal for each preconditions of the new steps. Using the same code as classical action insertion for all steps, all open conditions are provided.

Regarding the threats, we do the same operation for each link in the new plan than the one we do for new links. This guarantees that all threats are found in the new plan, therefore all classical flaws are discovered.

All new steps that are composite lead to an additional abstraction flaw so that all flaws are taken into account.

□

This proves that expansion does not introduce flaws that are not added in the POP agenda. Since POP must resolve all flaws in order to be successful and according to the proof of soundness of POP, HEART is sound as well.

Another proven property is that intermediary plans are valid in the classical definition of the term (without abstraction flaws) and that when using only this definition, HEART is sound on its anytime results too.

3.2 Completeness

The completeness of POP has been proven in the same paper than for its soundness [Erol et al., 1994]. This proof states that the way POP handles flaws makes it explore all possible solutions. Since our method uses the same exact algorithm only the differences must be proven to respect the contract met by the classical flaws.

Lemma (Expansion solves the abstraction flaw)**.** *The application of an expansion resolver, invalidates the related abstraction flaw.*

Proof. The expression of an abstraction flaw is the existence of its related composite step a in the plan. Since the application of an expansion resolver is of the form $S'_\pi = (S_\pi \setminus a) \cup S_m$ unless $a \in S_m$ (wich is forbiden by definition), $a \notin S'_\pi$.

□

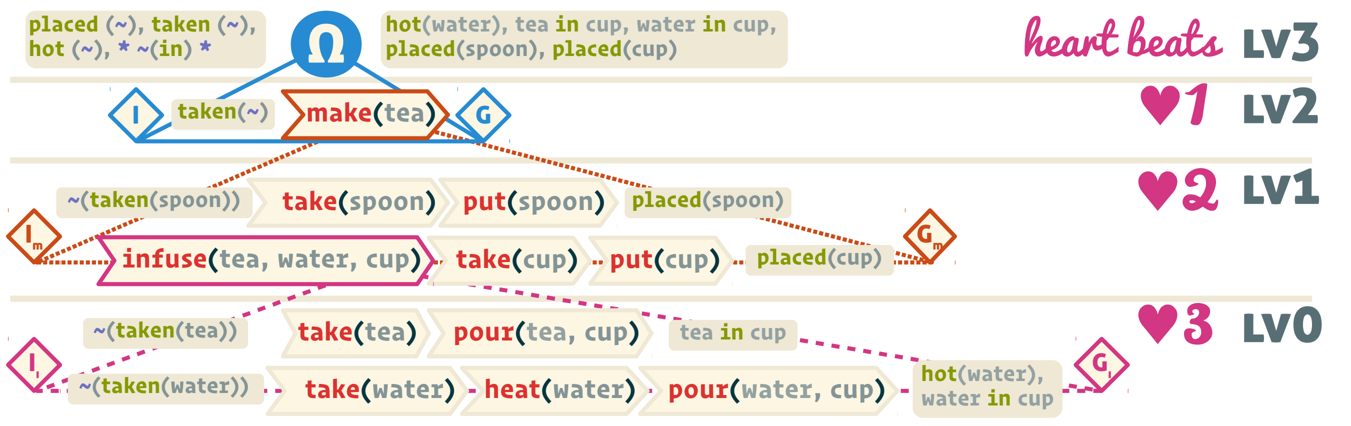

Figure 2: The heart beat process on the example domain. All actions that are not expanded durring a heart beat are kept in the next plan. Some details have been omitted in order to be concise.

Lemma (Solved abstraction flaw cannot reoccur)**.** *The application of an expansion resolver guarentees that $a \notin S_\pi$ for any other partial plan refined afterward without reverting the application of the resolver.*

Proof. By definition of the methods: $a \notin A^*_a$. This means that a cannot be introduced by it's expansion or the expansion of its proper actions.

HEART restricts the selection of providers for an open condition to operators that have an abstraction level strictly inferior to the one currently being made. So once a is expanded the current level is decreased and a cannot be selected by subgoal resolvers. It cannot either be contained in another action's methods that is selected afterward because otherwise by definition its level would be at least $lv(a) + 1$.

□

Since the implementation guarantees that the reversion is always done without side-effects, all the aspects of completeness of POP are preserved in HEART.

3.3 Computational profile

The complexity of HEART is very dependant on the domain's shape. We define the composition tree as having the empty root operator as its root, composite actions as node (their branches being all their proper actions) and the final leaves are atomic actions. The shape of the domain is the shape of that tree. Its two main metrics are its depth (abstraction level) and width. The **width** of a domain is the size of the biggest method : $width(\mathcal{D}) = \max_{m \in methods a}^{a \in O}(|S_m|)$.

It is common sense to expect real life domains to have a minimum width of 2. The worst case scenario is to have a domain of width 1 and with a very high abstraction level wich would give it the maximum amount of overhead for a domain that is equivalent to it's flat version. If we suppose that the shape of domains are homogenous (average width close to the maximum width)

FIXME law is more complicated than expected !

This means that computing the first level have a complexity that is close to being *linear* while computing the last level is of the same complexity as classical planning which is at least *P-SPACE* [Weld, 1994].

4 Results

In order to assess of its capabilities, our algorithm was tested on two different aspects : quality and complexity. All these tests were executed on an Intel® Core™ i7-7700HQ CPU clocked at 2.80GHz. The process used only one core and wasn't limited on time or memory. Each experiment was repeated between 10 000 and 700 times to ensure that variations in speed wasn't impacting the results.

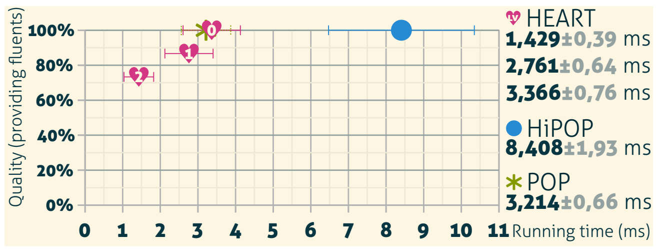

Figure 3: Evolution of the quality with computation time.

In figure 3, we show how the quality is affected by the abstraction in partial plans. The test is done on our example domain. The only variation when implementing our version of HiPOP was regarding the flaw selection and result representation, all the rest is identical. This shows that in some instances of hierarchical planning it may be more interesting to plan in a leveled fashion. It also exhibits a significative faster computation time for the first levels.

The quality is measured by counting the number of providing fluents in the plan $\left|\bigcup_{a \in S_\pi} eff(a)\right|$, which is actually used to compute the probability of a goal in intent recognition. We saw that a determining factor in speed for earlier levels was the number of preconditions of the composite actions. In these tests we didn't reduce that number in order to get the fairest results possible.

In the second test, we used generated domains. These domains are extremely simple. They consist of an action of level 5 that has a method containing a number of other actions of lower level. We call this number the width of the domain. Atomic actions are built with single fluent effects. The goal is the effect of the higher

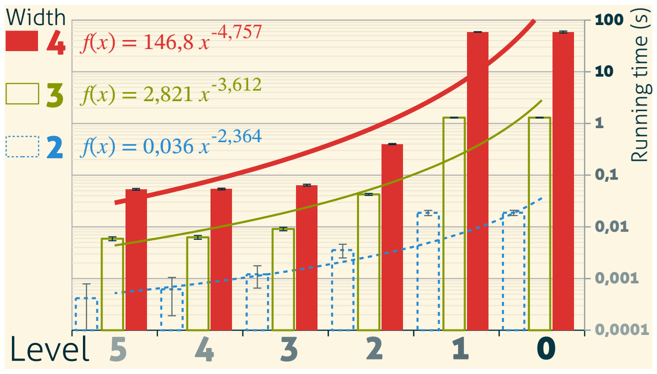

Figure 4: Impact of domain shape on the computation time by levels. The scale of the vertical axis is logarithmic.

level action and the initial state is empty. There are no negative fluents in these domains.

The figure 4 shows the computational profile of HEART for various levels and widths. It clearly behaves according to the exponantial law found earlier with exponant of the trend curve being very close to the actual width. This means that finding the first abstract plans is exponantially easier than to find the complete solution.

Conclusions

References

l

Baioletti, M., Marcugini, S., & Milani, A. (1998). Encoding planning constraints into partial order planning domains. In *KR* (pp. 608–616). MORGAN KAUFMANN PUBLISHERS.

Baker, C. L., Tenenbaum, J. B., & Saxe, R. R. (2007). Goal inference as inverse planning. In *CogSci*.

Bechon, P., Barbier, M., Infantes, G., Lesire, C., & Vidal, V. (2014). HiPOP: Hierarchical Partial-Order Planning. In *STAIRS* (Vol. 264, p. 51). IOS Press.

Chen, J., Chen, Y., Xu, Y., Huang, R., & Chen, Z. (2013). A Planning Approach to the Recognition of Multiple Goals. *IJIS*, *28*(3), 203–216.

Erol, K., Hendler, J. A., & Nau, D. S. (1994). UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *AIPS* (Vol. 94, pp. 249–254).

Fox, M., Long, D., & Magazzeni, D. (2017). Explainable Planning. *ArXiv Prepr. ArXiv170910256*.

Gerevini, A., Kuter, U., Nau, D. S., Saetti, A., & Waisbrot, N. (2008). Combining Domain-Independent Planning and HTN Planning: The Duet Planner. In *ECAI* (pp. 573–577).

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning: Theory & practice*. Elsevier.

Göbelbecker, M., Keller, T., Eyerich, P., Brenner, M., & Nebel, B. (2010). Coming Up With Good Excuses: What to do When no Plan Can be Found. In *ICAPS* (pp. 81–88).

Pellier, D., & Fiorino, H. (2007). A unified framework based on HTN and POP approaches for multi-agent planning. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (pp. 285–288). IEEE Computer Society.

Ramirez, M., & Geffner, H. (2009). Plan recognition as planning. In *IJCAI* (pp. 1778–1783).

Ramirez, M., & Geffner, H. (2010). Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*.

Sohrabi, S., Riabov, A. V., & Udrea, O. (2016). Plan Recognition as Planning Revisited. In *IJCAI*.

Talamadupula, K., Briggs, G., Chakraborti, T., Scheutz, M., & Kambhampati, S. (2014). Coordination in human-robot teams using mental modeling and plan recognition. In *IRoS* (pp. 2957–2962). IEEE.

Weld, D. S. (1994). An introduction to least commitment planning. *AI Mag.*, *15*(4), 27.