HEART: HiErarchical Abstraction for Real-Time Partial Order Causal Link Planning

Paper #8

Abstract

In recent years the ubiquity of artificial intelligence raised concerns among the uninitiated. The misunderstanding is further increased since most advances don't have explainable results. For automated planning, the research often target speed, quality, or expressivity. Most existing solutions focus on one area while not addressing the others. However, human-related applications require a complex combination of all those criteria at different levels. We present a new method to compromise on these aspects staying explainable. We aim to leave the range of potential applications as wide as possible but our main targets are human intent recognition and assistive robotics. The HEART planner is a real-time decompositional planner based on a hierarchical version of POCL. It cyclically explores the plan space while making sure that intermediary high level plans are valid and will return them as approximate solutions when interrupted. This paper aims to evaluate that process and its results related to classical approaches in terms of efficiency and quality.

Introduction

Since the early days of automated planning, a wide variety of approaches have been considered to solve diverse types of problems. They all range in expressivity, speed and reliability but often aim to excel in one of these domain. This leads to a polarisation of the solutions toward extremely specialised methods to tackle each problem. All of these approaches have been compared and discussed extensively in the books of Ghallab et al. (2004) and (2016).

Partially ordered approaches are popular for their least comitement aspect, flexibility and ability to modify plans using refinement operations (Weld 1994). This approach is often used in applications in robotics and collaborative planning. One of the most flexible approach is called **Partial Order Causal Link planning (POCL)**. It works by refining partial plans consisting of steps and causal links into a solution by solving all flaws compromising the validity of the plan.

Another approach is **Hierarchical Task Networks (HTN)** that are meant to tackle the problem using composite actions in order to define hierarchical tasks within the plan. Hierarchical domains are often considered easier to concieve and maintain by experts mainly because they seems closer to the way we think about these problems (Sacerdoti 1975).

In our work we aim to combine HTN and POCL in such a maner as to generate intermediary high level plans during the planning process. Combining these two approaches is not new (Young and Moore 1994; Kambhampati *et al.* 1998). Our work is based on **Hierarchical Partial Order Planning (HiPOP)** by Bechon *et al.* (2014). The idea is to expand the classical POCL algorithm with new flaws in order to make it compatible with HTN problems and allowing the production of abstract plans. In order to do this we present an upgraded planning framework that aims to simplify all notions to their minimal forms in order to reduce the size of the data structures to handle during runtime. We also propose some domain compilation techniques to reduce the work of the expert concieving the domain.

In all these works, only the final solution to the input problem is considered. That is a good approach to classical planning except when no solutions can be found. Our work focuses on the case when the solution couldn't be found in time or when high level explanations are preferrable to the complete implementation detail of the plan. This is done by focusing the planning effort toward finding intermediary abstract plans along the path to the complete solution.

In the rest of the paper, we detail how HiErarchical Abstraction for Real-Time (HEART) planner creates abstract intermediary plans that can be used for various applications. First, we discuss the motivations and related works to explain the choices behind our design process. Then we present the way we modelled an updated planning framework fitting our needs and then we explain our method and prove its properties to finally discuss the experimental results.

Motivations and Potential Applications

There are several reasons a problem might be unsolvable. The most obvious case is that no solution exists that meets the requirements of the problem. This has already been addressed by Göbelbecker *et al.* (2010) where "excuses" are being investigated as potential explanations for when a problem has no solutions.

Our approach tries to handle the case of when the problem is too difficult to solve within tight time constraints. For example in robotics, algorithms often need to be run within refresh rates of several Hertz giving the algorithm only fractions of a second to give an updated result. Since, planning has a PSPACE complexity (Erol *et al.* 1995) in its classical formalisation (it can be harder when functions, objects or quanitifiers are used), computing only the first plan level of a hierarchical domain is exponentially easier in relation to the complete problem.

While abstract plans are not complete solutions they still display a useful set of properties for various applications. The most imediate application is for explanable planning (Fox *et al.* 2017). Indeed a high level plan is more concice and does not contain unecessary implementation details that would confuse a non-expert.

Another potential application for such plans are relative to domains that works with approximative data. Our main example here is intent recognition which was the original motivation for this work. Planners are not meant to solve intent recognition problems. However, several works extended what is called in psychology the *theory of mind*. That theory is the equivalent of asking "*what would **I** do if I was them ?*" when observing the behavior of other agents. 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.* (2007) that use Bayesian planning to infer intentions. Ramirez and Geffner (2009) found an elegant way to transform a plan recognition problem into classical planning. This is done simply by encoding temporal coerol_complexity_1995nstraints in the planning domain in a similar way as Baioletti *et al.* (1998) describe it to match the observed action sequence. A cost comparison will then give a probability of the goal to be pursued given the observations. Chen *et al.* (2013) extended this with multi-goal recognition. A new method, proposed by Sohrabi *et al.* (2016), makes the recognition fluent centric. It assigns costs to missing or noisy observed fluents, which allows finer details and less preprocessing work than action based recognition. This method also uses a meta-goal that combines each possible goal and is realized when at least one of these goals is satisfied. Sohrabi *et al.* state that the quality of the recognition is directly linked to the properties of the generated plans. Thus guided diverse planning[1] was preferred along with the ability to infer several probable goals at once.

Related Works

Combining several planning paradigms is refered to as *hybrid planning*. HTN is often combined with classical approaches since it allows for a more natural expression of domains making expert knowledge easier to encode. These kind of planners are called **decompositional planners** (Fox 1997).

Most of the time the integration of HTN simply consists in calling another algorithm when introducing a composite operator during the planning process. In the case of the DUET planner by Gerevini *et al.* (2008), it is done by calling an instance of a HTN planner based on task insertion called SHOP2 [nau_shop2_2003] to deal with composite actions.

Some planners take the integration further by integrating the decomposition of composite action as a special step in their refinement process. Such works includes the discourse generation oriented DPOCL (Young and Moore 1994) and the paper by Kambhampati *et al.* (1998) generalizing the practice for decompositional planners.

In our case, we chose a planner called HiPOP concieved by Bechon *et al.* (2014) as our reference approach. The main difference here is that the decomposition is integrated into the classical POCL algorithm by only adding new types of flaws. This allows to keep all the flexibility and properties of POCL while adding the expresivity and abstraction capabilities of HTN. We also improved our own version of the planning framework used by HiPOP to reduce further the amount of changes needed to handle composite actions and to increase the efficiency of the resulting implementation.

As stated previously, our goal is to obtain intermediary abstract plans and to evaluate their properties. Another work have already been done on another aspect of those type of plans. The Angelic algorithm by Marthi *et al.* (2007) explicited the usefulness of such plans in the planning process itself and used them as heuritic guide. They also proven that,for a given fluent semantics, it is guaranteed that such abstract solutions can be refined into actual solutions. However, this work does not address the inherent properties of such abstract plans as approximate solutions and uses a more restrictive totally ordered framework.

Definitions

In order to make the notations used in the paper more understandable we gathered them in table 1. For domain and problem representation, we use a custom knowledge description language that is inspired from RDF and is based on triples and propositional logics. In that language quantifiers are used to quantify variables `*(x)` (forall x) but can also be simplified with an implicit form : `lost(~)` (nothing is lost). For reference the *exclusive quantifier* we introduced (noted `~`) is used for negation as well as the symbol for nil.

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

Domain

The domain specifies the allowed operators that can be used to plan and all the fluents they use as preconditions and effects.

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

[1] Diverse planning is set to find a set of m plans that are distant of d from one another.

Table 1: Our notations are adapted from Ghallab *et al.* (2004). The symbol ± demonstrates when the notation has signed variants.

Symbol	Description
$\mathcal{D}, \mathcal{P}$	Planning domain and problem.
$pre(a)$, $eff(a)$	Preconditions and effects of the action a.
$methods(a)$	Methods of the action a.
$\phi^{\pm}(l)$	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.
$L^{\pm}(a)$	Set of incoming (L^{-}) and outgoing (L^{+}) links of step a. No sign gives all adjacent links.
$a_s \xrightarrow{c} a_t$	Link with source a_s, target a_t and cause c.
$causes(l)$	Gives the causes of a causal link l.
$a_a \succ a_s$	A step a_a is anterior to the step a_s.
A_x^n	Proper actions set of x down n levels. A_x for $n = 1$ and A_x^* for $n = lv(x)$.
$lv(x)$	Abstraction level of the entity x.
$a \rhd^{\pm} a'$	Transpose the links of action a onto a'.
$l \downarrow a$	Link l participates in the partial support of step a.
$\pi \Downarrow a$	Plan π fully supports a.
$\ddagger_f\ a$	Subgoal : Fluent f isn't supported in step a.
$a_b \otimes l$	Threat : Breaker action a_b threatens causal link l.
$a \oplus_m$	Decomposition of composite action a using method m.
$var : exp$	The colon is a separator to be read as "such that".
$[exp]$	Iverson's brackets: 0 if $exp = false$, 1 otherwise.

- 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 example domain in listing 1 is inspired from the kitchen domain of Ramirez and Geffner (2010).

```
1 take(item) pre (taken(~), ?(item));
      //?(item) is used to make item into
      a variable.
2 take(item) eff (taken(item));
3 heat(thing) pre (~(hot(thing)),
      taken(thing));
4 heat(thing) eff (hot(thing));
5 pour(thing, into) pre (thing ~(in) into,
      taken(thing));
6 pour(thing, into) eff (thing in into);
7 put(utensil) pre (~(placed(utensil)),
      taken(utensil));
8 put(utensil) eff (placed(utensil),
      ~(taken(utensil)));
9 infuse(extract, liquid, container) ::
      Action; //Composite action of level 1
10 make(drink) :: Action; // Level 2
      containing infuse
```

Listing 1: Domain file used in our planner. In order to be concise the methods are omitted.

Definition 2 (Fluent)**.** A fluent f is a parameterized statement $r(arg_1, arg_2, ..., arg_n)$ where:

- r is a relation/function holding a property of the world.
- $arg_{i \in [1,n]}$ 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 do not use the closed world hypothesis: fluents are only satisfied when another compatible fluent is provided. Sets of fluents have a boolean value that equals the conjunction of all its fluents.

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

Definition 3 (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. S must contain an initial step I_π and goal step G_π.
- 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 its target a_t caused by the set of fluents c.

In HEART, *ordering constraints* are defined as the transitive cover of causal links over the set of steps. We note ordering constraints: $a_a \succ a_s$, 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: $a_a \neq a_s \wedge a_s \not\succ a_a$. In all plans, the initial and goal steps have their order guaranteed: $I_\pi \succ G\pi \wedge \nexists a_x \in S_\pi : a_x \succ I_\pi \vee G_\pi \succ a_x$. If we need to enforce order, we simply add a link without specifying a cause. The use of graphs and implicit order constraints help to simplify the model while maintaining its properties.

The central notion of planning is operators. Instanciated operators are usually called *actions*. In our framework, actions can be partially instantiated. We use the term action for both lifted and grounded operators.

Definition 4 (Action)**.** An action is a parametrized tuple $a(args) = \langle name, pre, eff, methods \rangle$ where:

- $name$ is the **name** of the action.
- pre and eff are sets of fluents that are respectively the **preconditions and the effects** of the action.
- $methods$ is a set of **methods** (partial order plans) that can realize the action. Methods, and the methods of their enclosed actions, cannot contain the parent action.

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

Composite actions are represented using methods. An action without methods is called *atomic*. It is of interest to note the divergence with classical HTN representation here since normally composite actions does not have preconditions nor effects. This is because in our case we will need to insert them into abstract plans.

In order to simplify the input of the domain, the causes in the methods are optional. In that case the causes are infered

by unifying the preconditions and effects with the same mechanism as in the subgoal resolution in our POCL algorithm. The preconditions and effects are also optional. When not provided, we use the following formula to compute the final preconditions and effects of a: $pre(a) = \bigcup_{a_s \in L^+(a)} causes(a_s)$ and $\mathit{eff}(a) = \bigcup_{a_s \in L^-(a)} causes(a_s)$. An instance of the classical POCL algorithm is then run on the result to ensure its coherence and the domain compilation fails if POCL cannot be completed or if nested methods contain their parent action as a step, since our decomposition hierarchy is acyclic ($a \notin A^*_a$, see definition 10).

Problem

Problem instances are often most simply described by two components: the initial state and the goal.

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 **root operator** of the problem which methods are potential solutions of the problem.

Example: We use a simple problem for our example domain. The initial state provides that nothing is ready, taken or hot and all containers are empty (all using quantifiers). The goal is to have tea made. For reference, listing 2 contains the problem instance we use as an example.

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

Listing 2: Example of a problem instance for the kitchen domain.

The root operator is initialized to $\Omega = \langle \text{""}, s_0, s^*, \{\pi_{lv(\Omega)}\} \rangle$, with s_0 being the initial state and s^* the goal specification. The method $\pi_{lv(\Omega)}$ is a partial order plan with the initial and goal steps linked together via Ω. The initial partial order plan is $\pi_{lv(\Omega)} = (\{I, G\}, \{I \xrightarrow{s_0} \Omega \xrightarrow{s^*} G\})$, with $I = \langle \text{"}init\text{"}, \varnothing, s_0, \varnothing \rangle$ and $G = \langle \text{"}goal\text{"}, s^*, \varnothing, \varnothing \rangle$.

Partial Order Causal Links

Our method is based on the classical POCL algorithm. It works by refining a partial plan into a solution by recursively removing all of its flaws.

Definition 6 (Flaws)**.** Flaws have a *proper fluent* f and a causing step often called the *needer* a_n. Flaws in a partial plan are either:

- **Subgoals**, *open conditions* that are yet to be supported by another step a_n often called *provider*. We note subgoals $\ddagger_f\ a_n$ (see definition 14).
- **Threats**, caused by steps that can break a causal link with their effects. They are called *breakers* of the threatened link. A step a_b threatens a causal link $l_t = a_p \xrightarrow{f} a_n$ if and only if $\neg f \in \mathit{eff}(a_b) \wedge a_b \not\succ a_p \wedge a_n \not\succ a_b$. Said otherwise, the breaker can cancel an effect of a providing step a_p, before it gets used by its needer a_n. We note threats $a_b \otimes l_t$.

Example: Our initial plan contains 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 to be fixed in order for the plan to be valid. In POCL it is done by finding their resolvers.

Definition 7 (Resolvers)**.** Classical resolvers are additional causal links that aim to fix a flaw.

- *For subgoals*, the resolvers are a set of potential causal links containing the proper fluent f in their causes while taking the needer step a_n as their target and a **provider** step a_p as their source.
- *For threats*, we usually consider only two resolvers: **demotion** ($a_b \succ a_p$) and **promotion** ($a_n \succ a_b$) of the breaker relative to the threatened link. We call the added causeless causal link a **guarding** link.

Example: The subgoal for $in(water, cup)$, 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.

The application of a resolver does not necessarily mean progress. It can have consequences that may require reverting its application in order to respect the backtracking of the POCL algorithm.

Definition 8 (Side effects)**.** Flaws that are caused by the application of a resolver are called *related flaws*. They are inserted into the *agenda*[2] 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 the deletion of a guarding link.

Flaws can also become irrelevant when a resolver is applied. It is always the case for the targeted flaw, but this 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 the removal of their needer.
- *Invalidated threats* happen when the breaker no longer threatens the causal link because the order guards the threatened causal link or either of them have been removed.

Example: Adding the action $pour(water, cup)$ causes a related subgoal for each of the preconditions of the action which are: the cup and the water must be taken and water must not already be in the cup.

[2] An agenda is a flaw container used for the flaw selection of POCL.

In algorithm 1 we present a generic version of POCL inspired by Ghallab *et al.* (2004, sec. 5.4.2).

Algorithm 1 Partial Order Planner

```
1  function POCL(Agenda a, Problem 𝒫)
2      if a = ∅ then                          ▷ Populated agenda of flaws needs to be provided
3          return Success                     ▷ Stops all recursion
4      Flaw f ← choose(a)                     ▷ Heuristically chosen flaw removed from agenda
5      Resolvers R ← solve(f, 𝒫)
6      for all r ∈ R do                       ▷ Non-deterministic choice operator
7          apply(r, π)                        ▷ Apply resolver to partial plan
8          Agenda a' ← update(a)
9          if POCL(a', 𝒫) = Success then      ▷ Refining recursively
10             return Success
11         revert(r, π)                       ▷ Failure, undo resolver application
12     a ← a ∪ {f}                            ▷ Flaw wasn't resolved
13     return Failure                         ▷ Revert to last non-deterministic choice
```

For our version of POCL we follow a refinement procedure that works in several generic steps. In figure 1 we detail the resolution of a subgoal as done in the algorithm 1.

The first is the search for resolver. It is often done in two separate steps : first select the candidates and then check each of them for validity. This is done using the the polymorphic function `solve` at line 5. In the case of subgoals, variable unification is performed to ensure the compatibility of the resolvers. Since this step is time consuming, the operator is instantiated accordingly at this step to factorise the computational effort. Composite operators have also all their methods instanciated at this step if they are selected as candidate. Then a resolver is picked non-deterministically for application (this can be heuristically driven). At line 7 the resolver is effectively applied to the current plan. All side effects and invalidations are handled durring the update of the agenda at line 8. If a problem occur, the algorithm backtracks and tries other resolvers. If no resolvers fits the flaw, the algorithm backtracks to previous resolver choices to explore all the possible plans and ensure completeness.

In definition 8, we mentioned effects that aren't present in classical POCL, namely *negative resolvers*. All classical resolvers only add steps and causal links to the partial plan. Our method needs to remove composite steps and their adjacent links when expanding them.

The Heart Of The Method

In this section, we explain how our method combines POCL with HTN planning and how they are used to generate intermediary abstract plans.

Additional Notions

In order to properly introduce the changes made for using HTN domains in POCL, we need to define a few notions.

Figure 1: Example of the refinement process for subgoal resolution

Definition 9 (Transposition)**.** In order to transpose the causal links of an action a' with the ones of an existing step a in a plan π, we do the following operation :

$$a \rhd^-_\pi a' = \left\{\phi^-(l) \xrightarrow{causes(l)} a' \,:\, l \in L^-_\pi(a)\right\} \cup (L_\pi \setminus L^-_\pi(a))$$

It is the same with $a' \xrightarrow{causes(l)} \phi^+(l)$ and L^+ for $a \rhd^+ a'$. This supposes that the respective preconditions and effects of a and a' are equivalent. When not signed, the transposition is generalized: $a \rhd a' = a \rhd^- a' \cup a \rhd^+ a'$.

Example: Using $a \rhd^- a'$ means that all incoming links of a are now incoming links of a' instead.

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 $A_a = \bigcup_{m \in methods(a)} S_m$.

  Recursively: $A^n_a = \bigcup_{a' \in A_a} A^{n-1}_{a'}$. For atomic actions $A_a = \emptyset$.

We note $A^*_a = A^{lv(a)}_a$ the set of **extended proper actions** of the action a.

Example: The proper actions of $make(drink)$ are the actions contained within its methods. The set of extended proper actions adds 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 express:[3]

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

Example: The abstraction level of any atomic action is 0 while it is 2 for the composite action $make(drink)$. The example domain (in listing 1) has an abstraction level of 3.

Abstraction In POCL

The most straightforward way to handle abstraction with POCL is illustrated in another planner called Duet (Gerevini *et al.* 2008) by managing hierarchical actions separately from a regular planner. We chose another way inspired by the works of Bechon *et al.* (2014) on a planner called HiPOP. The difference between the original HiPOP and our implementation of it is that we focus on the expressivity and the ways flaw selection can be exploited for partial resolution. Our version is lifted at runtime while the original is grounded for optimizations. All mechanisms we have implemented use POCL but with different management of flaws and resolvers. The original algorithm 1 is left untouched.

One of those changes is that resolver selection needs to be altered for subgoals. Indeed, as stated by the authors of HiPOP : the planner must ensure the selection of high level operators in order to benefit from the hierarchical aspect of the domain, overwise, adding operators only increases the branching factor.

We also need to add a way to deal with composite actions once inserted in the plan to reduce them to their atomic steps.

Definition 12 (Abstraction flaw)**.** It occurs when a partial plan contains a non-atomic step. This step is the needer a_n of the flaw. We note it $a_n\oplus$.

- *Resolvers*: An abstraction flaw is solved with an **decomposition 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 steps of the method as such: $a_n \rhd^- I_m \wedge a_n \rhd^+ G_m$, with $m \in methods(a_n)$.
- *Side effects*: An abstraction flaw can be related to the introduction of a composite action in the plan by any resolver 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 this composite step replaced by its method using an decomposition resolver. In order to the decomposition of a composite action into a plan, ll existing liks are reported to the initial and goal step of the selected method, while the composite action and its links are removed from the plan.

The main differences between HiPOP and HEART in our implementations are the functions of flaw selection and the handling of the results (one plan for HiPOP and a plan per cycle for HEART). In HiPOP, the flaw selection is made by prioritizing the abstraction flaws. Bechon *et al.* (2014) state that it makes the full resolution faster. However, it also loses opportunities to obtain abstract plans in the process.

[3]We use Iverson brackets here, see notations in table 1.

Cycles

The main focus of our work is toward obtaining **abstract plans** which are plans that are completed while still containing composite actions. In order to do that the flaw selection function will enforce cycles in the planning process.

Definition 13 (Cycle)**.** A cycle is planning phase defined as a tuple $c = \langle lv(c), agenda, \pi_{lv(c)}\rangle$ where : $lv(c)$ is the maximum abstraction level allowed for flaw selection in the $agenda$ of remaing flaws in partial plan $\pi_{lv(c)}$. Resolvers of subgoals are therefore constrained by the following: $a_p \downarrow_f a_n : lv(a_p) \leq lv(c)$.

During a cycle all abstraction flaws are delayed. 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 before the abstraction level is lowered for the next cycle: $lv(c') = lv(c) - 1$. Each cycle produces a more detailed abstract plan than the one before.

Abstract plans allow the planner to do an approximative form of anytime execution. At any given time the planner is able to return a fully supported plan. Before the first cycle, the plan returned is $\pi_{lv(\Omega)}$.

Example: In our case using the method of intent recognition of Sohrabi *et al.* (2016), we can already use $\pi_{lv(\Omega)}$ 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.

For each cycle c, a new plan $\pi_{lv(c)}$ is created as a new method of the root operator Ω. These intermediary plans are not solutions of the problem, nor do they mean that the problem is solvable. In order to find a solution, the HEART planner needs to reach the final cycle c_0 with an abstraction level $lv(c_0) = 0$. However, these plans can be used to derive meaning from the potential solution of the current problem and give a good approximation of the final result before its completion.

Example: In the figure 2, we illustrate the way our example problem is progressively solved. Before the first cycle c_2 all we have is the root operator and its plan π_3. Then within the first cycle, we select the composite action $make(tea)$ instanciated from the operator $make(drink)$ along with its methods. All related flaws are fixed until all that is left in the agenda is the abstract flaws. We save the partial plan π_2 for this cycle and expand $make(tea)$ into a copy of the current plan π_1 for the next cycle. The solution of the problem will be stored in π_0 once found.

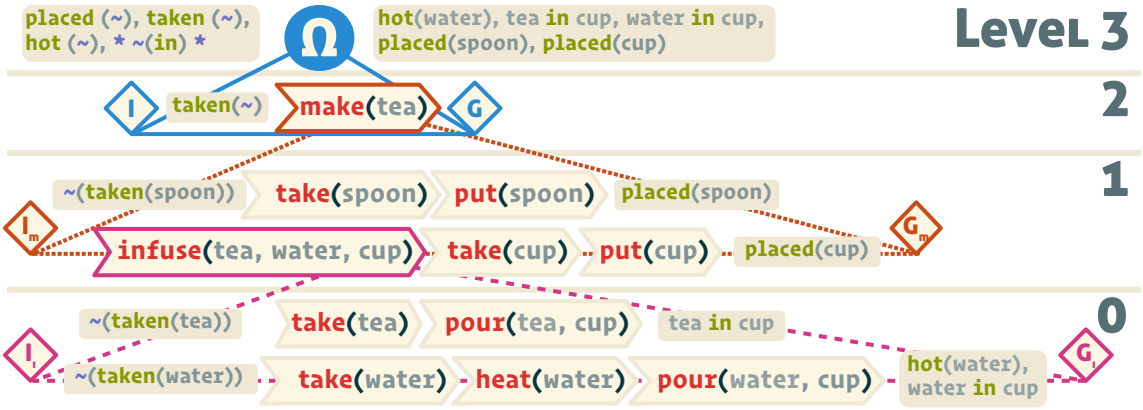

Figure 2: The cycle process on the example domain. Atomic actions that are copied from a cycle to the next are omitted.

Properties

Soundness

For an algorithm to be sound, it needs to provide only *valid* solutions. In order to prove soundness, we first need to define the notion of support.

Definition 14 (Support)**.** An open condition f of a step a is supported in a partial order plan π if and only if

$$\exists l \in L_\pi^-(a) \wedge \nexists a_b \in S_\pi : \\ f \in causes(l) \wedge (\phi^-(l) \succ a_b \succ a \wedge \neg f \in eff(a_b))$$

This means that the fluent is provided by a causal link and isn't threatened by another step. We note support $\pi \downarrow_f a$.

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 its logical equivalences for the proofs:

Definition 15 (Validity)**.** A plan π is a valid solution of a problem $\mathcal{P}$ if and only if $\forall a \in S_\pi : \pi \Downarrow a \wedge lv(a) = 0$.

We can now start to prove the soundness of our method. We base this proof upon the one done in (Penberthy *et al.* 1992). It states that for classical POCL 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 (Decomposition with an empty method)**.** *When a fully supported composite action $\pi \Downarrow a$ is expanded using an empty method $m = (\{I_m, G_m\}, \{I_m \rightarrow G_m\})$, adding all open conditions of G_m as subgoals (along with their related flaws) will result in a plan π', without any undiscovered flaws.*

Proof. The initial and goal steps of a method follows ($pre(a) = eff(a)$). By definition of full support:

$$L_{\pi'}^-(I_m) = L_\pi^-(a) \wedge pre(I_m) = pre(a) \implies (\pi \Downarrow a \equiv \pi' \Downarrow I_m)$$

The only remaining undiscovered open conditions in the plan π' are therefore those caused by G_m: $\{f \in pre(G_m) : \pi' \ddagger_f G_m\}$.

No new threats are introduced since the link between I_m and G_m is causeless and they inherit the order of the parent composite action. All added actions are atomic so no abstraction flaws are added.

□

Lemma (Decomposition with an arbitrary method)**.** *If a fully supported composite action $\pi \Downarrow a$ is replaced by an arbitrary method m and all open conditions contained within S_m as subgoals and all threatened links within L_m as threats are added to the agenda, the resulting plan π' will not have any undiscovered flaws.*

Proof. When replacing a composite action with a method in an existing plan we do the following operations: $S_{\pi'} = S_m \cup (S_\pi \setminus a)$, $L_{\pi'}(I_m) = a \triangleright I_m$. The only added flaws are then:

$$\bigcup_{a_m \in S_m}^{f \in pre(a_m)} \pi' \ddagger_f a_m \bigcup_{a_b \in S_{\pi'}}^{l \in L_{\pi'}} a_b \otimes l \bigcup_{a_c \in S_m}^{lv(a_c) \neq 0} a_c \oplus$$

That means that all added actions have their subgoals considered and all links have their threats taken into account along with all additional abstraction flaws.

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

□

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

Completeness

The completeness of POCL has been proven in the same paper as for its soundness (Penberthy *et al.* 1992). Since our method uses the same algorithm only the differences must be proven to

respect the contract of application and reversion. This contract states that applying a resolver prevents the reocurence of its flaw and that reverting the application of a resolver must restore the plan and agenda to their previous state.

Lemma (Decomposition solves the abstraction flaw)**.** *The application of an decomposition resolver invalidates the related abstraction flaw.*

Proof. Abstraction flaws arise from the existence of their related composite step a in the plan. Since the application of an decomposition resolver is of the form $S'_\pi = (S_\pi \setminus a) \cup S_m$ unless $a \in S_m$ (which is forbidden by definition 4), therefore $a \notin S'_\pi$.

□

Lemma (Solved abstraction flaws cannot reoccur)**.** *The application of an decomposition resolver on a plan π, guarantees that $a \notin S_\pi$ for any partial plan refined from π without reverting the application of the resolver.*

Proof. As stated in the definition of the methods (definition 4): $a \notin A^*_a$. This means that a cannot be introduced in the plan by its decomposition or the decomposition of its proper actions. Indeed, once a is expanded, the level of the following cycle $c_{lv(a)-1}$ prevents a to be selected by subgoal resolvers. It cannot either be contained in the methods of another action that are selected afterward because otherwise following definition 11 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 POCL are preserved in HEART.

□

Results

Experimental

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

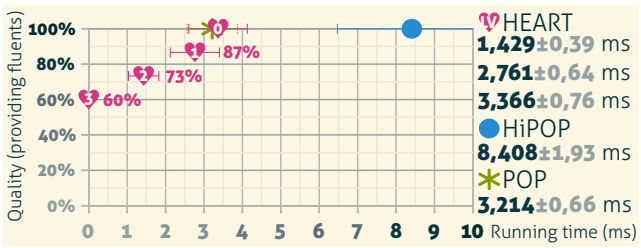

Figure 3: Evolution of the quality with computation time.

Figure 3 shows how the quality is affected by the abstraction in partial plans. The tests are made using our example domain (see listing 1). 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. The percentages are relative to the total number of unique fluents of the complete solution. These results show that in some cases it may be more interesting to plan in a leveled fashion to solve HTN problems. For the first cycle of level 3, the quality of the abstract plan is already of 60%. This is the quality of the exploitation of the plan *before any planning*. With almost 3 quarters of the final quality and less than half of the complete computation time, the result of the first cycle is a good quality/time compromise.

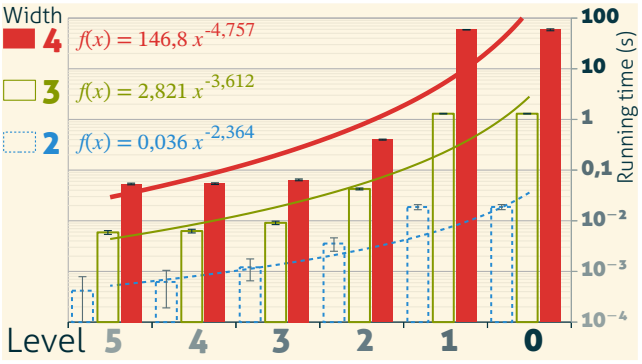

Figure 4: Impact of domain shape on the computation time by levels. The scale of the vertical axis is logarithmic. Equations are the definition of the trend curves.

In the second test, we used generated domains. These domains consist of an action of abstraction level 5. It has a single method containing a number of actions of level 4. We call this number the width of the domain. All needed actions are built recursively to form a tree shape. Atomic actions only have single fluent effects. The goal is the effect of the higher level action and the initial state is empty. These domains do not contain negative effects. Figure 4 shows the computational profile of HEART for various levels and widths. We note that the behavior of HEART seems to follow an exponential law with the negative exponent of the trend curves seemingly being correlated to the actual width. This means that computing the first cycles has a complexity that is close to being *linear* while computing the last cycles is of the same complexity as classical planning which is at least *P-SPACE* (depending on the details of the formalism) (Erol *et al.* 1995).

Conclusions

In this paper, we have presented a new planner called HEART based on POCL. An updated planning framework fitting the need for such a new approach was proposed. We showed how HEART performs compared to complete planners in terms of speed and quality. While the abstract plans generated during the planning process are not complete solutions, they are exponentially faster to generate while retaining significant quality over the final plans. By using these plans it is possible to find good approximations to intractable problems within tight time constraints.

References

Baioletti, M., S. Marcugini, and A. Milani
Encoding planning constraints into partial order planning domains, *International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers Inc., 1998, 608–616.

Baker, C. L., J. B. Tenenbaum, and R. R. Saxe
Goal inference as inverse planning, *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 29, 2007.

Bechon, P., M. Barbier, G. Infantes, C. Lesire, and V. Vidal
HiPOP: Hierarchical Partial-Order Planning, *European Starting AI Researcher Symposium*, IOS Press, 2014, 264,51–60.

Chen, J., Y. Chen, Y. Xu, R. Huang, and Z. Chen
A Planning Approach to the Recognition of Multiple Goals, *International Journal of Intelligent Systems*, 28 (3), 203–216, 2013. doi:10.1002/int.21565.

Erol, K., D. S. Nau, and V. S. Subrahmanian
Complexity, decidability and undecidability results for domain-independent planning, *Artificial intelligence*, 76 (1-2), 75–88, 1995.

Fox, M.
Natural hierarchical planning using operator decomposition, *European Conference on Planning*, Springer, 1997, 195–207.

Fox, M., D. Long, and D. Magazzeni
Explainable Planning, *Proceedings of IJCAI Workshop on Explainable AI*, Melbourne, Australia, August 2017.

Gerevini, A., U. Kuter, D. S. Nau, A. Saetti, and N. Waisbrot
Combining Domain-Independent Planning and HTN Planning: The Duet Planner., *Proceedings of the European Conference on Artificial Intelligence*, 2008, 18,573–577.

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

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

Göbelbecker, M., T. Keller, P. Eyerich, M. Brenner, and B. Nebel
Coming Up With Good Excuses: What to do When no Plan Can be Found., *Proceedings of the International Conference on International Conference on Automated Planning and Scheduling*, AAAI Press, May 2010, 20,81–88.

Kambhampati, S., A. Mali, and B. Srivastava
Hybrid planning for partially hierarchical domains, *AAAI/IAAI*, 1998, 882–888.

Marthi, B., S. J. Russell, and J. A. Wolfe
Angelic Semantics for High-Level Actions., *ICAPS*, 2007, 232–239.

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

Ramirez, M., and H. Geffner
Plan recognition as planning, *Proceedings of the International Conference on International Conference on Automated Planning and Scheduling*, AAAI Press, 2009, 19,1778–1783.

Ramirez, M., and H. Geffner
Probabilistic plan recognition using off-the-shelf classical planners, *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence*, 2010, 24,1121–1126.

Sacerdoti, E. D.
The nonlinear nature of plans, STANFORD RESEARCH INST MENLO PARK CA, 1975.

Sohrabi, S., A. V. Riabov, and O. Udrea
Plan Recognition as Planning Revisited, *Proceedings of the International Joint Conference on Artificial Intelligence*, Vol. 25, 2016.

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

Young, R. M., and J. D. Moore
DPOCL: A principled approach to discourse planning, *Proceedings of the Seventh International Workshop on Natural Language Generation*, Association for Computational Linguistics, 1994, 13–20.