COLOR : Abstract planning for assistive robotics (Draft)

Antoine Gréa

Introduction

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. It is hard to define an universal taxonomy for the domain. We can sort planners by their search space, their completeness, their use of uncertainty, their heuristics (that are extremely diverse), etc. **TODO: citation binge.** With all these colors of planners we yet wish to add a new one.

Let us introduce our planner named Composite Operators & LOcal Refinement (COLOR)[pun inteded].

Figure 1: A potential logo because this paper needs color

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 application [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.

The preferred type of planners used in this work is multiple planning (planning with a set of plans as a result) as it will return naturally the k best predictions **TODO: reword/introduce**. The paper also states that quality in the chosen planners is the principal criteria of selection. Since our applicative domain also include robotics, we wish to account also for the real time aspect of the prediction (a prediction is only useful *before* it happens).

This is the drive of the present paper. We therefore propose a planner with a focus on quality and anytime properties. It combines several planning approaches : it is based on Hierarchical Task Networks (HTN), Partial Order Planning (POP) and Local search heuris-

tics.

TODO: announce the superb plan of the paper

1 Related works

The sheer variety of planners out there makes dificult the act of comparing them. In order to siplify this task we propose a taxonomy of the deterministic planning domain. To our knowledge, no such taxonomy was proposed in recent works.

1.1 Taxonomy of deterministic planners

Figure 2: The taxonomy space that represents each characteristics of a deterministic planner

The figure 2 is a visual representation of the taxonomy we propose. The taxonomy is separated in six different *tracks*. Each track represent a category of characteristics. The characteristics closer to the center are considered to be lighter and faster than the one closer to the border that usually means more quality. In order to build this taxonomy we considered that every planning problems are explorational problems. Being inspired by the popular 6Ws method we simply asked the questions :

- *Where ?* **Search Space** : State based planners are often considered faster when plan based planners usually means more flexibility.

- *What ?* **Plan Type** : Most planners find a totally ordered sequence of actions. Some prefer to find a partially ordered plan. Both can be found in sets (what we call *multiple planning*)
- *Who ?* **Action Type** : The actors of planning are actions. The action model determines how expressive a solution plan will be but also will decrease its performance accordingly.
- *Why ?* **Guide Space** : A guide or heuristic is giving a notion of sens to planners. This category is so diverse (meta-heuristics, machine learning, user preferences, HTN, etc) that our solution was to reflect the search space and to differentiate with the space on wich the guide works on. **TODO : reword?**
- *hoW ?* **Building** : In this category we describe how a planner builds its plans. A simple directional building (forward or backward) is often faster than more complex ones (like plan re-use or repair).
- *When ?* **Timing** : The timing of the planning can affect how much time the process should take. More constraints on time means the planning process should be faster.

1.2 Analyse of related planners

In this section we will present several panners that influenced this paper and will position them on a miniature version of figure 2 using a radar chart representation.

1.2.1 SHOP & SHOP2

Figure 3

SHOP [7] is a HTN planner. It is working by fullfilling tasks instead of goals. Each task has a set of methods that can use other tasks in order to achieve their effects. It prefers more general operators durring the

planning process and outputs regular totally ordered sequences.

1.2.2 LPG and variations[1]

Figure 4

Gerevini's LPG [8] is a local search planner based on action graphs and constraints resoltuion. It started as GPG [9] and was featuring a simple Walksat local search on an levelled action graph. Each step is a refinement of a subgraph following a global dynamic heuristic based on cost assesment and Lagrange multipliers. Its performances and its ability to optimize and adapt plans made LPG a base planners of several other works.

1.2.2.1 LPG-Adapt

LPG-Adapt [10] is a variation of LPG used for plan repair and reuse. It is used mostly in dynamic environments in order to ensure that reused plans are not altered too much from iteration to iteration. This stability is an interesting property for online planning and give a valuable metric for diverse planning methods.

1.2.2.2 LPG-d

LPG was also adapted for diverse planning. LPG-d [11] is a diverse planner that uses LPG's capacity to be driven by action costs in order to account for user preferences. It is derived from a an improved version of LPG [12] that already can do planning with multi-criteria for plan quality.

[1]The taxonomy figure for LPG contains in dark blue the taxonomy for classical LPG and in grey the taxonomy for its variations.

1.2.2.3 Duet

The Duet planner [13] is Gereveni's approach to HTN planning. It is a modified version of LPG that has an additional constraint violation type. This constraint violation aims to resolve abstract actions inserted with HTN planning by invoking SHOP2 [14] on the considered part of the problem.

1.2.3 HiPOP : not a musical

Figure 5

Despite its name HiPOP has little to do with music and more with automated planning and HTN. HiPOP [15] is a planner that uses Partial Order Planning (POP) techniques with HTN planning. It adds upon recent planners such as VHPOP [16] with HTN based heuristics. The classical POP flaws must have been completed to adapt to abstract actions and how they introduce invisible flaws.

1.3 Positioning of COLOR

Figure 6

COLOR is a HTN diverse online planner made for inverted planning. It aims to use local search and plan refining techniques for fast and incomplete planning. The goal is to be able to obtain in a given time the most precise plans possible.

Table 1: Most used symbols in the paper. **TODO: Update that**

Symbol	Description
$pre(o)$, $eff(o)$	Preconditions and effects of the operator o
$\mathcal{D}$	Planning domain
$\mathcal{P}$	Planning problem
$l_{\rightarrow}$, $l_{\leftarrow}$	Source and target of the causal link l
$o_1 \succ o_2$	Precedence operator (o_1 precedes o_2)
O^{Π}	Operator graph of the set of operators O
$d_{\pi}^{\pm}(o)$	Outgoing and incoming degrees of o
$d^{\pm}(o)$	Proper degrees of o ($\|pre(o)\|$ and $\|eff(o)\|$)
$L_{\pi}^{\pm}(o)$	Outgoing and incoming causal links of o
$C(\pi)$	Set of cycles in partial plan π
$C_o(\pi)$	Set of cycles in π which o is part of
$C_o^s(\pi)$	$\{o\}$ if o has a self cycle in π, $\emptyset$ otherwise
$\mathcal{F}^{\pm}(\pi)$	Set of flaws in π
$r(f)$	Resolvers of the flaw f
$f.n$	Needer of the flaw f
$f(\pi)$	Application of the flaw f on plan π
$\Downarrow \pi$	Full support of π
$\pi \models \mathcal{P}$	The partial plan π is a valid solution of $\mathcal{P}$

2 Definitions

We aim to define all necessary notions in order to make a framework in which we build our planner. This framework is based on reusability and recursive notions. Such properties make further definition simpler and factorize the work as we only need to define future notions for a reduced set of cases.

2.1 Notations

In this paper, we use the notation defined in table 1. Our notation is adapted from the one used in [17].We use the symbol $\pm$ to signify that there is a notation for the positive and negative symbols but the current formula works regardless of the sign. All related notions will be defined later.

2.2 Domain definitions

With the aim to respect the constraints of our application field, we need to define our own planning framework. We chose a level of expressiveness similar to that of PDDL 3.1 with object-fluents support [18]. This particular type of domain description was chosen because values are more meaningful and therefore changing them makes repairs on plans easier.

Definition 1 (Domain)**.** Our domain definition takes into acount some modifications necessary to achieve the type of planning we aim for. A domain is a tuple $\mathcal{D} = \langle T, C_{\mathcal{D}}, R, F, O \rangle$ where

- T is the set of **types** used in the domain for variables and constants. We provide types with a relation of **subsumption** noted $t_1 \prec t_2$ with $t_1, t_2 \in T$ meaning that all instances of t_1 are also instances of t_2.
- $C_{\mathcal{D}}$ is the set of typed **domain specific constants**.
- R is the set of **relations** (also called properties). They are used to express aspects and properties of the world. We note them as a tuple $r = \langle name, r_s, r_t \rangle$ with $name$ as the relation's name, r_s and r_t as the source and target type and arity (ex.: $Object^2$ signify a pair of objects). When a relation is instanciated it is called a *fluent* and binds an object to n subjet(s) (n being the arity of the relation).
- F is the set of possible **fluents**. An additional unary relation is needed on the fluents : $\neg f$ is used to signify that the fluent is negative. We also add special behavior to sets of fluents. For exemple we note $F^{\pm}$ the set of positive/negative fluent present in F. A set of fluents can be either seen as a *state*, *effects* or *preconditions* :
  - *Preconditions* have a predicate of validity noted $s \models p : [(p^+ \in s) \land (p^- \notin s)]$, meaning that all positive fluents must be present and all negative fluents absent in the state s for the precondition p to be valid.
  - *Effects* have an application operation noted $s + e = (s \setminus e^-) \cup e^+$. The values erasive, meaning that if a fluent is present with different target value the value in the state s is erased by the value in e except if the target arrity of the relation of the fluent is infinite. **TODO: example to explain maybe + open/closed world hypothesis**
- O is the set of **operators**. An operator is a tu-

ple $o = \langle name, pre, eff, methods, lv \rangle$ with *name* as the operator's name, *pre* as the set of fluent coresponding to a conjonction that must be met to apply the operator using the set of fluent that represents the effect : eff. We note *methods* is the set of *methods* of the operator that contains a partial plan to realise the operator. The set can be empty in witch case the operator is said to be *atomic*. *l* is the abstraction level of the operator.

This definition lays some important bases of our framework. Our operators are **composite** meaning that a *method* must be chosen and instanciated as a partial plan in order to complete any plan containing an instanciation of this operator.

We note $lv_{max}(\mathcal{D})$ the maximum abstraction level of a given domain.

2.3 Partial plan definitions

Definition 2 (Partial Plan)**.** We define a partial plan as a tuple $\langle S, L, B \rangle$ with :

- S the set of **steps**. A step is a partially or fully instantiated operators also called *actions*. It has two unary relations :
    - $lv_i(a)$ whitch affects to each step an interval of integers that corresponds to the **abstraction levels of the step**.
    - $dummy(a)$ is a predicate that gives true iff the step is a **dummy action**. Dummy actions are always linked to their parent via a *hierarchical causal link* and cannot be removed without removing the parent action.
- L the set of **causal links**.
- B the set of **binding constraints**.

Partial plans can be seen as a simple form of planning graphs. It is a directed acyclic graph with action as vertices and causal links as edges. In the case of HTN we add another dimensionality to the planning graph with *abstraction levels*. On the highest level we use the most generic composite operators. The last level is for atomic actions belonging to the most specific composite operators. Each level can be seen as a

plan by itself. We introduce the set of **hierarchical causal links** that is a subset of $L_H \subset L$. These links are not to be acounted for support (see definition 3). Another propriety of these levels is that $lv(o) <= lv_i(a)$ with a being an instanciation of o. We can also extract regular partial plans from leveled partial plans at any level. π_n is a restriction of the leveled partial plan π at level n.

We note $l : a_s \xrightarrow{c} a_t$ the **causal link** l that binds the *source step* a_s to the *target step* a_t using the *cause c*. A cause is a simple set of fluents that is provided by the source to the target.

Definition 3 (Support)**.** We define support as a property of preconditions, steps and partial plans. A partial plan π is fully supported if each of its steps $o \in S$ is fully supported. A step is fully supported if each of its preconditions $f \in pre(o)$ is supported. A precondition is fully supported if there exists a causal link l that provides it. We note full support of a partial plan as :

$$\pi \Downarrow \equiv \begin{array}{l} \forall a \in S \; \forall f \in pre(a) \; \exists l \in L_\pi^-(a) \setminus L_H : \\ (f \in l \wedge \; \nexists t \in S(l_\rightarrow \succ t \succ a \wedge \neg f \in eff(t))) \end{array}$$

with $L_\pi^-(a)$ being the incoming causal links of a in π and $l_\rightarrow$ being the source of the link.

This means that in order to be fully supported a partial plan needs to have all its step's precondition supplied with causal links without possible order that allows a step to threaten another link. We add the notation $\pi \downarrow$ for partial support, meaning that some preconditions might not be supported. Obviously preconditions cannot be partially supported.

A specificity of our definition is that the ordering constraints are part of causal links. When an ordering constraint doesn't have a cause we use a **bare causal links** (mostly used for threats). We also introduce the **precedence operator** noted $a_i \succ a_j$ with $a_i, a_j \in S$ iff there is a path of causal links that connects a_i to a_j. We call a_i *anterior* to a_j.

The notion of support can be extended to a single level $\pi \Downarrow^n$

2.4 We have a 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 **root operator** which methods correspond to the solution of the problem. Its level is set to $lv_{max}(\mathcal{D}) + 1$.

The root operator contains the **initial state and goal** (noted respectively I and G) of the problem. In the context of multiple planning, this operator may have several methods that corresponds to several result plans.

2.5 Plan refinement definitions

Partial Order Planning fixes flaws in a partial plan to refine it into a valid plan that is a solution to the given problem. Since we work with a HTN approach the classical flaws must be adapted [15]. Therefore, we present the updated flaws for our version of POP.
Definition 5 (Subgoal). A flaw in a partial plan, called subgoal, is a missing causal link required to support a precondition of a step. We can note a subgoal as $a_n \nvDash_s$ with s being **proper fluent** of the subgoal (also called *open condition*) and also a unsuported precondition of the action a_n called the **needer** of the subgoal.

It is to be noted that hierarchical causal links does not account for support and therefore doesn't solve subgoals. Also flaws can only be applied at a given abstraction level.
Definition 6 (Threat). A flaw in a partial plan called threat consists of having an effect of a step that can be inserted between two actions with a causal link that is threatened by the said effect. We say that a step a_b is threatening a causal link $a_p \xrightarrow{t} a_n$ iff $a_b \neq a_p \neq a_n \land \neg t \in eff(a_b) \land a_p \succ a_b \succ a_n$ with a_b being the **breaker**, a_n the *needer* and a_p a *provider* of the *proper fluent t*.

Flaws are fixed via the application of a resolver to the plan. A flaw can have several resolvers that match its needs.
Definition 7 (Resolvers). A resolver is a potential causal link defined as a tuple $r = \langle a_s, a_t, f \rangle$ with:

- $a_s, a_t \in S$ being the source and the target of the resolver,
- f being the considered fluent.

For classical flaws, the resolvers are simple to find. For a *subgoal* the resolvers are sets of the potential causal links between a possible provider (along allowed operators or steps) of the proper fluent and the needer. To solve a *threat* there are mainly two resolvers: a causal link between the needer and the breaker called **demotion** or a causal link between the breaker and the provider called **promotion**.

Once the resolver is applied, the algorithm needs to take into account **side effects** of that application.
Definition 8 (Side effects). Flaws that arise because of the application of a resolver on the partial plan are called causal side effects or *related flaws*.

We can derive this definition for subgoals and threats:

- **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 causal link.

3 Algorithm

The algorithm of COLOR works in several phases. It first need to compile the domain in a particular way. Then it will plan on each abstraction level and do some additional processing between each levels.

3.1 Domain compilation

This is the first phase. It is running when processing files describing the planning domain. It contains several mecanisms that are added to the standard domain compilation.

3.1.1 Classical mecanisms

The standard compilation mecanisms are straightforward. First the software will parse the files into the correct memory representation of the domain. While doing so it checks some basic sytax and semantic constraints. It also check for inconsistancies in the domain (action with contradicting effects, non realisable preconditions) and either remove them or throw an error. It also prepare the instanciation inertia in order to quicken the planning procedure.

The mecanism searches for operator that matches

$$o \in O \wedge \left[\begin{array}{l} pre(o) \cap eff(o) \neq \emptyset \vee \\ \exists \{f, \neg f\} \subset pre(o) \vee \exists \{f, \neg f\} \subset eff(o) \end{array} \right]$$

3.1.2 Method binding constraints

From there, the process do the basic intertia on each methods of abstract operators. This inertia is used to give an application formula that gives the instanciation constraints regarding variable affectation and typing. This formula is cached for the planning process to access. **TODO : How to compute that exactly, see Damien**

3.1.3 Abstraction level computation

In order for the algorithm to work, abstract operators need an appropriately set abstraction level.
Definition 9 (Abstraction level)**.** The abstraction level of an operator is equal to the highest abstraction level used in its methods plus one. We can note : $lv(o) = 1 + \{\max(lv(o_{sub}), 0) | o_{sub} \in m_o \in methods(o)\}$

In order to find this level for every operator, the algorithm needs two more steps :

- Looping on every operators to find cycles in the dependencies of abstract operators. The cycles happens when an operator uses in its method a parent operator. The cycles are cut ate the weakest point (smallest operator/method)

- Recursively increasing abstraction levels after initialization at 1 on all operators. This step stops once the levels stay stable. **TODO : maybe better way to do this but whatever ?**

3.2 Online phase

This phase starts when the problem is first provided.

3.2.1 Instanciation inertia

Instanciation inertia is imediatly started, begining by the highest level operators. The intertia can remove methods right away as their instanciation constraints might be impossible in the context of the given problem. This will reduce greatly the branching factor and therefore quicken the planning process. In order to do this we use methods described in [19]. **TODO: see with Damien about that**

3.2.2 Levelled planning

The algorithm for each level is the same. It is a variation of classical POP with LPG like heuristics. The local search is done at the method level (between the start and end dummy actions). If not in a method the scope starts at the last end dummy action and ends at the next start dummy action.

If we are planning at abstraction level n, the we can only instanciate operators of level n or above. This is done to restrict the search space according to user's intent [20]. Obviously, using standard POP algorithm the planning will quickly fail if these high level operators can't achieve the goal. This failure is part of the extended algorithm.

When a subgoal can't find a resolver in the set of available operators it backtracks in a special way. The flaw is discarded in a separated agenda and the planning continues. Once there is no more flaws or the planning process efectively fails, all flaws are gathered along with the current state of the partial plan.

3.2.3 Changing level

Once a level finishes its partial plans found are copied into the next level, namely $n - 1$. For each plan the algorithm takes all the abstract actions and replace it with instanciations of its available methods framed by dummy start and end actions. All incoming causal links are copied to the start action and the outgoing ones to the end one. The algorithm also adds hierarchical causal links to the parent action.

One can note that unallowed action can happens in the methods. These are allowed only when changing level and are then behaving like normal steps of the partial plan. Flaws can use such a step even if the related operator is forbiden.

3.2.4 Finalization

Once the level 1 is reached, all operators becomes available to use and the planning reaches completude. The formal fail and success of the algorithm happens only at that level. If the algorithm succeeds, all found plans are processed by an upward propagation of actions. If an action doesn't have a parent, it is propagated in the superior levels with their causal links (adapted for parent actions if needed). The result is a set of consistent plan across all levels.

The advantages of this is that each plan can be used as "hints" for intent recognition. **TODO: Next part on properties + more formalization**

4 Notes

Justifying the property by saying that the exact description of a problem is imposible because too precise. Each level down adds a lot to the complexity without helping much the meaning. We should just stick to a *density* of information and define saturation levels.

The problem with most systems AI is semantic reasoning. They can't produce explanations of their logic. They can't explain a result and recognize incoherences.

References

[1] C. L. Baker, J. B. Tenenbaum, and R. R. Saxe, "Goal inference as inverse planning," in *Proceedings of the 29th annual meeting of the cognitive science society*, 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 *PRINCIPLES OF KNOWL-EDGE REPRESENTATION AND REASONING-INTERNATIONAL CONFERENCE-*, 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," *International Journal of Intelligent Systems*, 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 *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, 2014, pp. 2957–2962.

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

[7] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila, "SHOP: Simple hierarchical ordered planner," in *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, 1999, pp. 968–973.

[8] A. Gerevini and I. Serina, "LPG : A Planner Based on Local Search for Planning Graphs with Action Costs." in *AIPS*, 2002, vol. 2, pp. 281–290.

[9] A. Gerevini and I. Serina, "Lagrange multipliers for local search on planning graphs," in *Workshop on Local Search for Planning and Scheduling*, 2000, pp. 37–54.

[10] M. Fox, A. Gerevini, D. Long, and I. Serina, "Plan Stability: Replanning versus Plan Repair." in *ICAPS*,

2006, vol. 6, pp. 212–221.

[11] T. A. Nguyen, M. Do, A. E. Gerevini, I. Serina, B. Srivastava, and S. Kambhampati, "Generating diverse plans to handle unknown and partially known user preferences," *Artificial Intelligence*, vol. 190, pp. 1–31, 2012.

[12] A. E. Gerevini, A. Saetti, and I. Serina, "An approach to efficient planning with numerical fluents and multi-criteria plan quality," *Artificial Intelligence*, vol. 172, no. 8, pp. 899–944, 2008.

[13] A. Gerevini, U. Kuter, D. S. Nau, A. Saetti, and N. Waisbrot, "Combining Domain-Independent Planning and HTN Planning: The Duet Planner." in *ECAI*, 2008, pp. 573–577.

[14] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN planning system," *J. Artif. Intell. Res.(JAIR)*, vol. 20, pp. 379–404, 2003.

[15] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal, "HiPOP: Hierarchical Partial-Order Planning," in *STAIRS 2014: Proceedings of the 7th European Starting AI Researcher Symposium*, 2014, vol. 264, p. 51.

[16] H. akan L. Younes and R. G. Simmons, "VHPOP : Versatile heuristic partial order planner," *JAIR*, pp. 405–430, 2003.

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

[18] D. L. Kovacs, "BNF definition of PDDL 3.1," *Unpublished manuscript from the IPC-2011 website*, 2011.

[19] A. Ramoul, D. Pellier, H. Fiorino, and S. Pesty, "Une approche totalement instanciée pour la planification HTN," 2016.

[20] S. Kambhampati, A. Mali, and B. Srivastava, "Hybrid planning for partially hierarchical domains," in *AAAI/IAAI*, 1998, pp. 882–888.