HTN Planning Approach Using Fully Instantiated Problems

Abdeldjalil Ramoul, Damien Pellier, Humbert Fiorino, Sylvie Pesty

HTN Planning Approach Using Fully Instantiated Problems

Abdeldjalil Ramoul*§, Damien Pellier§, Humbert Fiorino§ and Sylvie Pesty§
§ Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France
firstname.lastname@imag.fr
* Cloud Temple
215 Avenue Georges Clemenceau, 92024 Nanterre Cedex
https://www.cloud-temple.com/

***Abstract*—Many planning techniques have been developed to allow autonomous systems to act and make decisions based on their perceptions of the environment. Among these techniques, HTN (*Hierarchical Task Network*) planning is one of the most used in practice. Unlike classical approaches of planning. HTN operates by decomposing task into sub-tasks until each of these sub-tasks can be achieved an action. This hierarchical representation provide a richer representation of planning problems and allows to better guide the plan search and provides more knowledge to the underlying algorithms. In this paper, we propose a new approach of HTN planning in which, as in conventional planning, we instantiate all planning operators before starting the search process. This approach has proven its effectiveness in classical planning and is necessary for the development of effective heuristics and encoding planning problems in other formalism such as CSP or SAT. The instantiation is actually used by most modern planners but has never been applied in an HTN based planning framework. We present in this article a generic instantiation algorithm which implements many simplification techniques to reduce the process complexity inspired from those used in classical planning. Finally we present some results obtained from an experimentation on a range of problems used in the international planning competitions with a modified version of SHOP planner using fully instantiated problems.**

I. Introduction

Act and make rational decisions based on perceptions of the environment is a central issue in intelligent and autonomous systems. Many techniques were developed to handle this issue by trying to make the decision process computable. Among these techniques, HTN (*Hierarchical Task Network*) planning is one of the most used in practice [1]–[3]. HTN planners are more used in industrial applications for efficiency reasons and also for the expressiveness of HTN languages that allow to specify in the planning domains some experts knowledge that can be used to improve the performance of planning algorithms. Unlike conventional planning [4] where the goal is defined as a set of propositions to achieve. In HTN planning, the goal is expressed as a task or a set of tasks to achieve called *Goal Tasks* and to which it is possible to associate constraints. A couple (constraints, tasks) is called a *Task Network*. The search for a solution plan consists in decomposing the goal tasks complying with their defined constraints to sub-tasks until reaching a set of primitive sub-tasks that can be executed by actions in the classical sense of planning. This recursive decomposition is performed by applying rules defined in hierarchical planning operators named *Methods*. Each method defines a possible decomposition of a task into a set of sub-tasks with their associated constraints. The decomposition ends when the decomposition process results in a task network containing only primitive tasks achievable by actions and whose all associated constraints are verified.

HTN planners can be divided into two categories [5]. This division is based on the nature of the research space used by these algorithms: Plan space based and state space algorithms. In the first category, the search space consists in several task networks representing partial plans in which planners do not maintain state during the research. At each stage of the search process the task network obtained after a decomposition is considered as a partial plan with constraints that must be satisfied in the following decomposition. The plan based representation of search space allows to get partial order plans. There is many plan based planners, like *NOAH (Nets Of Action Hierarchies)* [6], [7], *Nonlin (Non-Linear Planner)* [8], [9] , *O-Plan* [10] and *O-Plan2* [11], *SIPE (System for Interactive Planning and Execution)* [12] and *SIPE-2* [13].In 1994, the algorithm *UMCP (Universal Method Composition Planner)* [14] was the first algorithm whose correctness and completeness were proved. In the second category, planners maintain states during search process. Each task network contains a representation of the state in addition to the tasks and the constraints which are most of the time only preconditions. decomposition process consists of choosing the tasks to decompose following the execution order, then apply the decomposition method in which the preconditions are verified in the task network state. Can be cited chronologically as planner in this category, *SHOP* [15], *SHOP2* [16] and *SIADEX* [17].

Alongside the development of HTN planning, many efficient planning algorithms without a hierarchical representation were developed, like *Fast Forward* [18] or *Fast Downward* [19], [20]. These algorithms perform a preprocessing step that consists in enumerating and instantiating all the possible actions from the operators defined in the planning problem. This step is crucial for these algorithms for several reasons. First of all, this enumeration or instantiation step reduces the number of actions in the problem through different simplification mechanisms. This has the consequence of reducing the branching coefficient of the research space. Secondly, generating all possible actions in a planning problem allows to perform a priori study on the reachable propositions in the problem. This study is a necessary prerequisite for the preparation and the development of efficient heuristics to guide search process [18], [21]–[25]. Thirdly, This preprocessing step is a necessary prerequisite for encoding planning problems in many formalism such as CSP [26], [27] or SAT [28]–[31]. However, To our knowledge, this preprocessing was never implemented and adapted in an HTN context. For all these reasons, it would be interesting to perform the instantiation and the simplification of planning operators, by integrating the hierarchical dimension, to reduce the number of possible decomposition and the complexity of the research space.

Basing on the observation that HTN planning is an extension of classical planning that adds decomposition methods to define high level knowledge. We propose in this paper an instantiation and simplification algorithm for HTN problems. This algorithm instantiate operators using mechanisms developed for the instantiation of classical planning problems and extends them for the instantiation and the simplification of HTN tasks and decomposition methods. We start by defining the HTN formalism. Thereafter, we introduce the instantiation problem of HTN planning and make explicit its complexity. After that, we detail the instantiation and the simplification mechanisms proposed for HTN planning problems and present a modified version of the algorithm SHOP that we have named *iSHOP (instantiated SHOP)* which takes as input a fully instantiated problem. Finally, we show how the instantiation algorithm improves performance of HTN algorithms in terms of research time by comparing iSHOP with classical SHOP.

II. HTN FORMALISM

Before introducing the instantiation process and its complexity, we begin at first by defining an HTN problem and presenting the basic concepts defined by the HTN formalism. All the given examples in this section and the rest of this paper are based on the *rover* domain used in the third and the fifth international planning competition.

Definition 2.1: An HTN problem is a 4-tuple *P = (s_0, T, O, M)* where s_0 is the initial state defined by a set of propositions characterizing the world, *T* is a set of initial tasks which defines the goal, *O* is a set of operators defining the actions that can be achieved, and *M* is a set of methods defining the possible decomposition of a composed task.

Definition 2.2: An operator is a 3-tuple *o = (name(o), pre(o), eff(o)). name (o)* is a syntactic expression of the form *t(u_1, ..., u_k)* where *t* is the operator name and u_1, ...,u_k its parameters. *Pre(o)* and *eff(o)* are logical expressions defining respectively the preconditions that must be verified to apply the operator and the effects that define the properties generated by the operator. $pre^+(o)$, $pre^-(o)$, $eff^+(o)$, $eff^-(op)$ are their positive and negative subsets.

An action a is a totally instantiated operator that defines a transition function allowing to pass from a state s to a state s' as follows: $s' = ((s \setminus eff^-(a)) \cup eff^+(a))$. a is applicable in a state s if $pre^+(a) \subseteq s$ and $pre^-(a) \cap s = \emptyset$. The preconditions and the effects of an action contains *propositions* that are fully instantiated atomic formula.

Definition 2.3: A method is a 3-tuple *m = (name (m), subtasks(m), constr (m)). name (o)* is a syntactic expression of the form *t(u_1, ..., u_k)* where *t* is the name of the method and u_1,....,u_k are its parameters. *Subtasks (m)* is a set of tasks composing *t(m)* and *Constr(m)* is a set of constraints on *subtasks(m)*.

Every constraint represents a condition which must be verified in all solution plans. Four types of constraints can be defined in a method:

- *Order constraint*: An order constraint is an expression of the form *(series $t_1, t_2, ..., t_k$)*. It means that in a solution plan, the task t_1 Must be ordered before the task t_2, t_2 before t_3 so on to t_k.
- *Before constraint* : A *Before* constraint is an expression of the form *(before (φ) ($t_1, ..., t_k$))* where φ is a logical expression, and ($t_1, ..., t_k$) is a list of tasks. *Before* constraints are used to verify that the expression φ is true in the state that happens just before the first task in the group ($t_1, ..., t_k$).
- *After constraint* : As *Before* constraints, *After* constraints are expression of the form *(after (φ) ($t_1, ..., t_k$))*. They mean that φ Must be true in the state resulting from the execution of the last task in ($t_1, ..., t_k$).
- A *Between* constraint is an expression of the form *(between (φ) ($t_{11}, ..., t_{1k}$) ($t_{21}, ..., t_{2k}$))*. The logical expression φ must be true in all stats between the last task of ($t_{11}, ..., t_{1k}$) and the first of ($t_{21}, ..., t_{2k}$).

Definition 2.4: A task is an expression of the form *t(u_1, ..., u_k)*, where *t* is the name of the task and u_1,...,u_k its parameters. A method or an operator are considered as *relevant* for a task *t* if *t(u_1, ..., u_k)* is equal to *name(m)* or *name(o)*. There are two types of tasks: (1) Composed tasks that can be decomposed into sub-tasks by applying a relevant method, (2) primitive tasks that are defined by operators and that cannot be decomposed.

As example, the action (*navigate rover waypoint3 waypoint2*) from the *rover* domain is an instantiated version of the primitive task (*navigate ?r ?w1 ?w1*). *Navigate* is the name of the task and *(?r ?w1 ?w2)* its parameters. (*rover, waypoint3, waypoint2*) are constants representing the parameters of the instantiated task. We can cite as example of a composed task in the Rover domain (*get_soil_data ?waypoint*) which can be decomposed into: navigation, soil sampling and data transmission sub-tasks.

III. INSTANTIATION PROBLEM

The instantiation process consists of replacing all typed variables by constants of the same type. The instantiation process generates all possible instances with all possible combinations of constants. The instantiation problem depends on the number of parameters in the method or the operator and also on the domain cardinality of each variable. The complexity of instantiating one operator with k parameters x_i

where $i \in \{1, ..., k\}$ having as domain $D(x_i)$ with a cardinality $|D(x_i)|$ is equal to:

$$\mathcal{O} = \prod_{i=0}^{k} |D(x_i)|$$

With a big number of constants, the number of generated instances rapidly increases which can lead an explosion of the instantiation and the search complexity. Let us take as example the instantiation of the *rover* domain with the problem *p40* from the IPC-5. The number of instances obtained just from the instantiation of the operator *communicate_soil_data (?x - rover ?l - lander ?p1 - waypoint ?p2 - waypoint ?p3 - waypoint)* is 14 million = $(14 \times rover) \times (1 \times lander) \times (100 \times waypoint1) \times (100 \times waypoint2) \times (100 \times waypoint3)$.

In HTN, the problem of the complexity is particularly important as it concerns in addition to the operators all the methods defined in the problem. We know that generally the methods have a parameter count more important than the operators, then the complexity of methods instantiation is bigger than operators one. However, the difference between the number of methods instances and the count of operators instances depends on how the planning problem is defined. In the instantiation of the problem *p40* of the *rover* domain, the method *send_soil_data (?x - rover ?from - waypoint)* has only two declared parameters. However, it contains two tasks, namely: *(do_navigate ?x ?w1)* and *(communicate_soil_data ?x ?l ?from ?w1 ?w2)* which introduces three additional parameters. In this example, the number of generated instances is 14 millions instances which is equal to the instances of the operator *communicate_soil_data*. But we can easily imagine other tasks and additional parameters leading to more instances.

IV. Instantiation algorithm for HTN problems

To deal with the complexity of instantiating, the instantiation process is realized through several enumeration and simplification steps that allow to only generate the actions, methods and propositions that are relevant in the problem. We rely on the work done in [32] and on the notion of *inertia* introduced by [33]. In this section, we are going to explain the instantiation process of an HTN problem in its two phases: (1) Operators instantiation phase and (2) methods instantiation phase.

A. Operators Instantiation and Simplification Process

The instantiation process does not just generate all possible operator instances. It preforms also many simplifications to reduce the complexity of the instantiated problem. The process of operators instantiation passes through four stages: A logical expressions normalization step, An operators instantiation step, a logical expressions simplification step and finally a step of simplification and reduction of operators.

1) Normalization of logical expression: In this step, all logical expressions containing implications and quantifiers are reformulated into a conjunctive or disjunctive normal form which is a prerequisite of any manipulation of logical expressions. All the logical formulas contained in the operators are affected by this rewriting.

2) Instantiation of operators: Instantiating an operator consists of replacing every declared variable by the corresponding constants. For every combination of values, a new action is created. The example 4.1 shows an instance of the operator *navigate* from the *rover* domain where the variable *?x* was replaced by the constant *rover1*, the variable *?p1* by *waypoint3* and the variable *?p2* by *waypoint2*. The instantiation uses the variables types to find corresponding constants. However in some cases, the problems are not typed. It is thus necessary to deduce the types of variables from the unary predicates before instantiating the operators. The inference process of the operators variables types is presented in detail in [33].

Example 4.1:

```
(:action navigate
 :parameters (rover1 waypoint3 waypoint2)
 :precondition
   (and
     (available rover1) (at rover1 waypoint3)
     (can_traverse rover1 waypoint3 waypoint2)
     (visible waypoint3 waypoint2))
 :effect
   (and (not(at rover1 waypoint3))
        (at rover1 waypoint2)))
```

3) Simplification of atomic formula: The simplification phase needs to be done as early as possible during the instantiation to optimize the process and reduce its cost. The simplification consists of evaluating the atomic formula contained in the operators to $true$ or $false$ using the concept of inertia.

- **A positive inertia** is a proposition that never appears in the positive effects of an operator, therefore it is never produced by an action of the problem. If a proposition is a positive inertia and it is not in the initial state, it will never be in any state of the problem.
- **A negative inertia** is a proposition that never appears in the negative effects of an operator, therefore it is never consumed by an action of the problem. Consequently, if the propositions considered as negative inertia are in the initial state, they will be true in all states of the problem.

The computing of inertia sets is done in one single pass on the set of operators. The predicates that are not in the inertia set are called "*fluent*" and may appear or disappear from one state to another. If p is a totally instantiated proposition and I the initial state, the simplification of operators atomic formulas is made following these rules:

- **If** p is a positive inertia and $p \notin I$ **then** p is simplified to **false**.
- **If** p is a negative inertia and $p \in I$ **then** p is simplified to **true**.
- **Else** p cannot be simplified.

All the simplified propositions can be deleted from the problem and all those which are not simplified are considered as relevant. In the example 4.2, the preconditions of the action *navigate* are simplified to *false* if the proposition *(can_traverse* *rover1 waypoint3 waypoint2)* is in the set of positive inertia and is not in the initial state.

Example 4.2:

```
:precondition
  (and
    (available rover1) (at rover1 waypoint3)
    (can_traverse rover1 waypoint3 waypoint2)
    (visible waypoint3 waypoint2))
:precondition false
```

4) Simplification of actions: The goal is to find and remove actions that can never be applied in the problem basing on the atomic simplifications presented before. As mentioned, the atomic expressions can be simplified to $true$ or $false$ what allows to simplify the preconditions and the effects by applying logical transformation rules. If these simplifications allow to reduce to *true* or *false* all the logical expression contained in the preconditions or the effects of an action, it can be simplified as follows:

- If the precondition or the effect of an action is replaced by $false$, the action is deleted from the planning problem. If the precondition is false, the action can never be applied. In the case where the effect is simplified to *false*, the application of the action produces an inconsistent state.
- If all the action effects are evaluated to $true$, the action can be removed from the problem because it does not produce any change.

B. Methods Instantiation and Simplification Process

The methods instantiation has five stages: a step of normalization performed on the logical expressions, a step of types inference, a step of methods instantiation, a step of logical expressions simplifications, and finally, a step of simplification and reduction of the instantiated methods.

1) Normalization of Logical expressions: In HTN, the normalization of logical expressions transforms all logical formula contained in methods constraints into a conjunctive normal form for future manipulations.

2) Inference of variable types: In methods definition, variables can be used in sub-tasks and constraints without being declared in the methods parameters. These variables have no defined type, what requires to infer their types before being able to perform the methods instantiation. The inference process for each undeclared variable is done in two stages:

a) Inferring types from sub-tasks:

- Get T the set of sub-tasks containing in their parameters the undeclared variable.
- for each sub-task $t \in T$, get the operator op_r or the methods m_r relevant for t.
- for each sub-task $t \in T$, get the declared types in the parameters of op_r or m_r. If the result is two types A and B, where B is a sub-type of A, keep the type B.
- If several types having no inheritance link are recovered, then an error must be reported.

b) Inferring types from constraints:

- Get *P* the set of propositions used in the constraints and containing the undeclared variable in their parameters.
- For each proposition $p \in P$, get the undeclared variable types from the relevant atomic formula.
- If several types that have no inheritance link are obtained, then an error must be reported. Otherwise, if the types have an inheritance link, keep the type with no sub-type.

Finally, if several types are obtained from the various inference stages ,it is necessary to keep the type with no sub-types. If there are still several types, it signifies an error.

3) Instantiation of methods: Like in operators instantiation, the instantiation of a method consists of replacing its variables by all corresponding constants. Every new method instance corresponds to a combination of constants affected to declared and undeclared variables.

Example 4.3:

```
(:method do_navigate
 :parameters   (rover1 waypoint3 waypoint0)
 :expansion
  ((tag t1 (navigate rover1 waypoint3 waypoint1))
   (tag t2 (visit waypoint1))
   (tag t3 (do_navigate rover1 waypoint1 waypoint0))
   (tag t4 (unvisit waypoint1)))
 :constraints
  (and
   (series t1 t2 t3 t4)
   (before (and
            (not(can_traverse rover1 waypoint3
                 waypoint0))
            (not(visited waypoint1))
   (between (visited waypoint1) t2 t4)))
```

The example 4.3 shows an instance of the method *do_navigate* from the *rover* domain with the following combination: *?x* $\equiv$ *rover1*, *?from* $\equiv$ *waypoint3*, *?to* $\equiv$ *waypoint0*, *?mid* $\equiv$ *waypoint1*. the value affected to *?mid* corresponds to the type *waypoint*, basing on the inference of types.

4) Simplification of atomic formula: The simplification aims to evaluate the atomic formula contained in the methods constraints to *true* or *false* using the concept of inertia. The example 4.4 shows the simplification of the atomic formula contained in the constraint before defined in the method *do_navigate*. Knowing that the proposition *(can_traverse rover1 waypoint3 waypoint0)* is a negative inertia, if it is declared in the initial state, it will be simplified to *true* and all the logical expression will be simplified to *false*.

Example 4.4:

```
(before (and (not(can_traverse rover1 waypoint3
    waypoint0))
    (not(visited waypoint1))) t1)
(before (false) t1))
```

We can clearly see that the atomic formula simplification process decreases considerably the complexity of the constraints and consequently reduces the complexity of the processing done in the methods simplification step.

5) Simplification of methods: The methods simplification aims to identify and delete the methods containing constraints that can never be verified in the problem. Two kinds of simplification are realized on one single pass on the methods set:

a) Constraints based simplification: The simplification based on the constraints relies on the evaluations of their logical expressions which can be simplified in $true$ or $false$ in the previous step. The simplification is done basing on the following rules:

- If the logical expression is simplified as $true$, the constraint is removed from the method, as it is always verified.
- If the logical formula of a constraint is simplified to $false$, the whole method is deleted. In this case, this constraint can never be verified in the problem, which means that this method will never lead to a solution plan.

b) Tasks based simplification: The simplification based on the tasks aims to delete methods containing primitive tasks that cannot be applied. Assuming that the operators simplification is made before the methods one, the simplification procedure is performed as follows:

1) Get T, the set of primitive tasks defined in the method to be simplified,
2) For every task $t \in T$, check if the relevant action for t was deleted during the operators simplification phase. If it is the case, then the whole method is deleted. Otherwise, it is preserved.

Another simplification of methods based on composed tasks could be performed. In this simplification, a method is removed if it contains a composed task that has no relevant method. This happens if all the relevant methods for a task are removed by the simplification process performed before. The composed task based simplification would require many iterations because each method simplification could simplify all the relevant methods for a task and involves a new iteration. The process continues until the stabilization of the methods set. This technique is under development and was not included in the performed experimentation.

V. Tests and results

In the previous section, we formally defined the instantiation and simplification algorithm for HTN problems. the goal of the next section is to show if reducing the number of methods and operators using instantiation improves the performance of HTN algorithms. To answer to this question, we implemented a simplified version of *SHOP* algorithm named *iSHOP (instanciated SHOP)* which takes as input a fully instantiated HTN problem. After that, we compared the two algorithms basing on processing time and plans lengths.

A. iSHOP algorithm

iSHOP algorithm is coded in JAVA using the PDDL4J planning library [34]. This library includes required modules for classical planning lexical and syntactical analysis. We developed and included in it the HTN lexical and syntactical analysis module and the HTN instantiation and simplification tool.

Algorithm 1 presents an iSHOP generic process that takes as input the problem *(S,T,O,M,P)* where S is a state, T =

Algorithm 1 iSHOP(S, T, O, M, P)

```
 1: if $T = \emptyset$ then
 2:     return empty plan
 3: end if
 4: $t \leftarrow$ The first task in $T$
 5: $U \leftarrow T - t$
 6: if $t$ is primitive then
 7:     $a \leftarrow$ the relevant action for $t$
 8:     if $precond(a) \in S$ then
 9:         $P \leftarrow$ iSHOP($a(S), U, O, M, P$)
10:         return $P$
11:     else
12:         return failure
13:     end if
14: else
15:     $active \leftarrow \{m \in M$ and $m$ is relevant for $t\}$
16:     if $active \neq \emptyset$ then
17:         Nondeterministically choose $m \in active$
18:         if $precond(m) \in S$ then
19:             $T' \leftarrow U \cup tasks(m)$
20:             iSHOP($S, T', O, M, P$)
21:         else
22:             return failure
23:         end if
24:     else
25:         return failure
26:     end if
27: end if
```

$(t_1, t_2, ..., t_n)$ is a task list, *O* the set of actions, *M* the instantiated methods and *P* the propositions defined in the problem. The algorithm starts by checking the set of tasks *T* and returns *an empty plan* if *T* is empty. Else, the relevant action or methods are tested and applied depending on if the first task is primitive or composed. The process is repeated until all the tasks in the node are processed. The resulting plan is made of all applied actions in the node during the process. Unlike SHOP, no instantiation constraint is used and preconditions checking is performed with a simple inclusion test on bit sets representing the problem state. The representation of state in the form of bit sets is only possible through the instantiation process that enumerate all propositions, the operators and the methods in the problem.

B. Experimental framework

In our comparison between SHOP and iSHOP, we made a first set of tests where we compared the execution performance of both algorithms on three planning domains: rover, childsnack and satellite. Each domain was tested with at least 20 problems focusing on the most complex ones. The first test phase was limited to the three problems mentioned above, mainly because of the difficulty of defining HTN methods in two different languages, one for SHOP and one for iSHOP, while ensuring that the two definitions are as close as possible in order to not distort the comparison. In addition to the domains, we rewrote in an HTN representation all the problems given in the planning competition for the three domains, allowing us to compare also the two algorithms with *Fast Downward* planner [19] which is a classical planning algorithm which has participated in planning competitions. To reproduce exactly the same conditions for both algorithms, we wanted to have both of them coded in the same language. So we chose to compare iSHOP which is coded in Java with a planner developed within the team and implementing SHOP in Java. We did not use JSHOP2 planner available on the internet as it implements the algorithm SHOP2 and shows runtime errors on the rover domain.

1) Hardware: All tests were performed on a computer with a multi-core intel Core i7 clocked at 2.2 GHZ and 16GB DDR3 RAM with 1600MHZ. The number of CPU cores does not affect the results because both algorithms does not implement parallelization techniques and the java process is executed on one core. The other CPU cores are used by the JVM only for data compression if the 10GB memory allocated to the process is exceeded.

2) Comparison criteria: The main purpose of instantiation is to reduce the complexity of planning problems and search time. Therefore, we compared the algorithms basing on criteria used in *agile* track of the international planning competition. In this track, each planner gets a score for each problem based on its own search time compared with other algorithms best time. For iSHOP we considered total time composed of pre-processing time and search time. We compared also the algorithms on plans lengths, but it is strongly related to domains definitions and not really relevant particularly between HTN and classical planners.

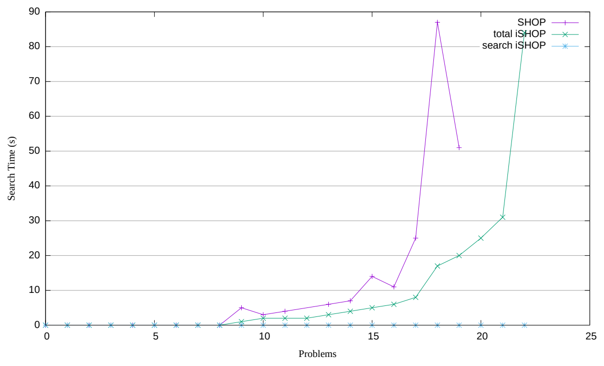
The figure 1 shows search times obtained during the first set of test with iSHOP, SHOP and Fast Downward planners. On the X-axis are represented the three domains planning problems and on Y-axis is represented (in seconds) the processing time to find the first solution. If there is no result displayed for a problems it means that the planner was not able to find a solution in the allotted time which is equal to 10 minutes. The figure 2 shows the results of the algorithms in term of plans lengths which are represented as the number of actions on the Y-axis.

C. Evaluation and results

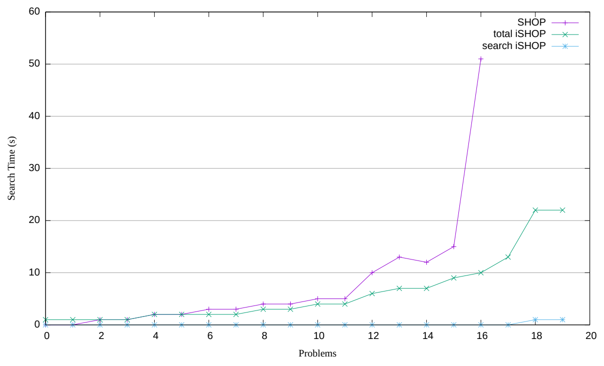
All results obtained with an HTN planning algorithms are relative to the way that the domain was written. Taking into account this characteristic and trying to be as fair as possible, we coded very similar HTN domain to test both algorithms. However, slight differences could not be avoided, this is mainly due to the specificities of each planner input languages.

Considering search times displayed in figure 1 (a, b, and c), we observe that iSHOP takes less time to find a solution than SHOP. The difference in search time grows with problems complexity. The difference is very small or nil in small problems and about several tens of seconds in complex problems. We can observe also that SHOP is not able to find a solution within the allotted time from the 19th problem in rover domain, from the 16th in childsnack and from the 12th in satellite, where iSHOP finds solutions for all problems without exceeding the maximum allowed time.

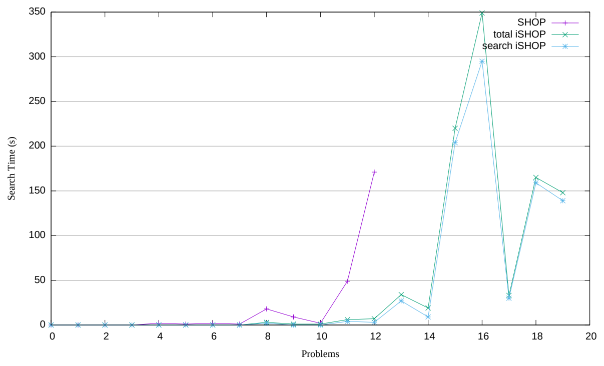
We can notice that the pre-processing time represents more than 90% of iSHOP's total processing time in childsnack and rover, but less that 10% in satellite. This is due to the way that methods are defined and the inertia present in the problem.



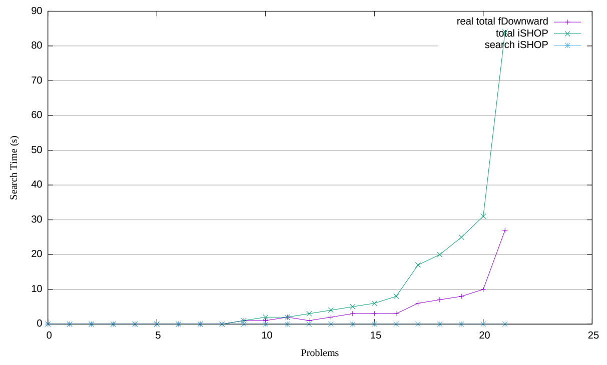
(a) Processing Times of SHOP and iSHOP on Rover Domain



(b) Processing Times of SHOP and iSHOP on Childsnack Domain

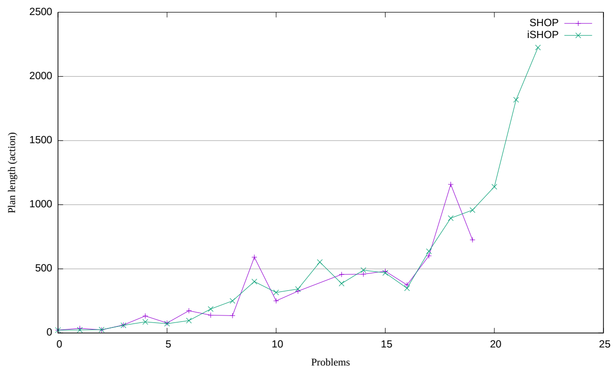


(c) Processing Times of SHOP and iSHOP on Satellite Domain

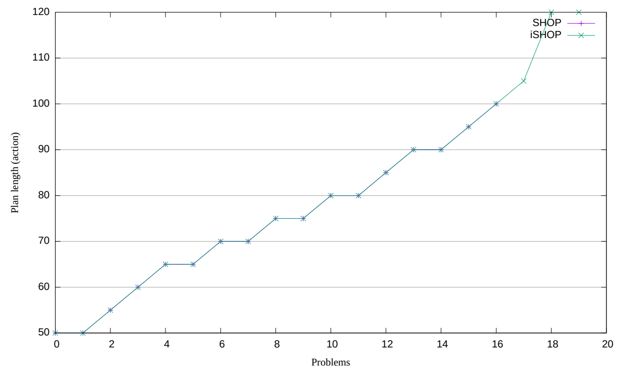


(d) Processing Times of Fast Downward and iSHOP on Rover Domain

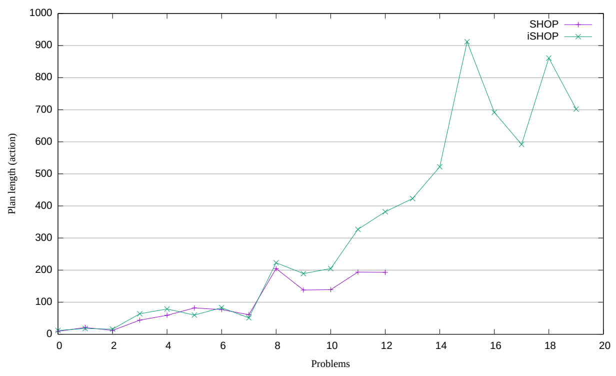
Fig. 1. Processing Times Comparison Between iSHOP, Fast Downward and SHOP on The Three Planning Domains : Rover, Childsnack and Satellite



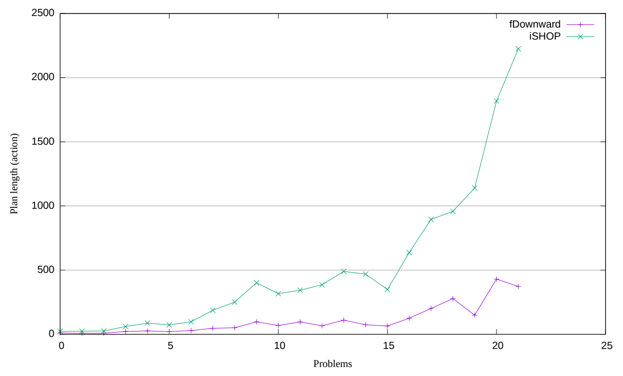
(a) Plans Lengths of SHOP and iSHOP on Rover Domain



(b) Plans Lengths of SHOP and iSHOP on Childsnack Domain



(c) Plans Lengths of SHOP and iSHOP on Satellite Domain



(d) Plans Lengths of Fast Downward and iSHOP on Rover Domain

Fig. 2. Plans Lengths Comparison Between iSHOP, Fast Downward and SHOP on the Three Planning Domains : Rover, Childsnack and Satellite

In rover for example, we followed the domain defined as benchmark in SHOP but we would have been able to define the domain in another way, with more constraints, which would have resulted in more simplifications. In figure 1(d), the bad performance of iShop compared to Fast Downward is mainly due to the pre-processing times for the same reasons mentioned above. Without pre-processing time, iSHOP's search time is equal to 0.78 seconds with less than 7 844 explored nodes on the most complex problem in rover, versus 8.21 seconds search time and 7 091 explored nodes with Fast Downward.

The table 1 shows the scores obtained by the three tested algorithms on the three planning domains relying on the IPC-8 agile track rules. We notice that on the rover domain Fast Downward gets a perfect score of 22/22. iSHOP is second with a score of 18.93/22. So iSHOP is 13.95% less efficient than Fast Down ward and 16.4% more efficient that SHOP. On childsnack domain, iSHOP gets the best score with 18.59/20 and is respectively 62.5% and 20.85% more efficient than Fast Downward and SHOP. On satellite domain, Fast Downward gets a score of 18.65/20 where iSHOP has only 17.28/20, so iSHOP is 6.23% less efficient than Fast Downward and 43% more efficient than SHOP. The overall score on the three domains is 45.5/62 for Fast Downward, 54.81/62 for iSHOP and 37.57/62 for SHOP. On the three domains, iSHOP is 15% more efficient than Fast Downward and 27.8% more efficient than SHOP. All the presented results show that iShop is more efficient in terms of processing time than the classical version

TABLE I
Scores of the Algorithms Fast Downward, iSHOP and SHOP

	rover			childsnack			satellite		
Pb	fdwd	iSHOP	SHOP	fdwd	iSHOP	SHOP	fdwd	iSHOP	SHOP
00	1	1	1	0	0,88	1	1	1	1
01	1	1	1	0,49	0,91	1	1	1	1
02	1	1	1	0,34	0,94	1	1	1	1
03	1	1	1	1	0,61	0,61	1	1	1
04	1	1	1	1	0,64	0,66	1	1	0,33
05	1	1	1	1	0,70	0,72	1	1	0,39
06	1	1	1	0	1	0,95	1	1	0,37
07	1	1	1	0	1	0,95	1	1	0,37
08	1	1	1	0	1	0,91	1	0,43	0,32
09	1	0,88	0,63	1	0,88	0,84	1	0,66	0,42
10	1	0,86	0,79	0	1	0,92	0,86	1	0,76
11	1	0,96	0,81	0	1	0,93	1	0,61	0,39
12	1	0,72	0,63	0	1	0,83	0,91	1	0,42
13	1	0,77	0,64	0	1	0,81	0,98	1	0
14	1	0,81	0,59	0	1	0,83	0,88	1	0
15	1	0,74	0,62	0	1	0,81	1	0,90	0
16	1	0,73	0,54	0	1	0,59	0	1	0
17	1	0,70	0,47	0	1	0	1	0,48	0
18	1	0,70	0,54	0	1	0	1	0,69	0
19	1	0,67	0	0	1	0	1	0,47	0
20	1	0,67	0						
21	1	0,67	0						
	22	18,93	15,32	4,84	18,59	14,42	18,65	17,28	7,81

of SHOP with a clear advantage on complex problems.

In HTN, the length of plans depends greatly on decomposition methods defined in the problems. The figure 2 (a,b,c) shows the lengths of plans produced by iSHOP and SHOP. We see clearly that both algorithms generates plans with very similar lengths. But in figure 2 (d), The length of the plan generated by Fast Downward is significantly smaller than that obtained with iSHOP. This is mainly due to the HTN approach of iSHOP that defines the goal as a succession of ordered tasks, while Fast Downward, in addition to very powerful heuristics, defines the goal as a state, where the order of actions is not restricted by the definition of the problem.

VI. Conclusion

We have presented in this paper a new fully instantiated approach for HTN planning. The peculiarity of this approach is that it reuses the instantiation and simplification methods used in classical planning, and proposes new rules for instantiating HTN methods. We have used a formalism with a high level of expressiveness for methods definition allowing to define domains that can be used by all HTN planners, either by state based or plan based partial order planers. We aimed to demonstrate the effectiveness of our approach on three planning domains through the implementation and the testing of the iSHOP algorithm. The experimentation results show that a fully instantiated HTN planning approach allows to obtain much shorter search times than a classical HTN approach. The advantage of the instantiated approach does not only lie in enhancing processing time, since it allows to perform reachability studies using completely instantiated methods and allows to implement and use search heuristics. Given the obtained results during the first phase of testing, we plan to continue to do more tests on other planning domains, to have more comparison points and confirm the current results. We also plan in future works to propose and implement search heuristics with iSHOP algorithm and another HTN plan based algorithm which is under development within the team and test the performance of the instantiated HTN approach with these heuristic on both types of algorithms.

References

[1] M. Weser, D. Off, and J. Zhang, "HTN robot planning in partially observable dynamic environments," in *Proceeding of the International Conference on Robotics and Automation*. IEEE, 2010, pp. 1505–1510.

[2] G. Bevacqua, J. Cacace, A. Finzi, and V. Lippiello, "Mixed-initiative planning and execution for multiple drones in search and rescue missions." in *Proceeding of the International Conference on Automated Planning and Scheduling*, 2015, pp. 315–323.

[3] R. Strenzke and A. Schulte, "The MMP: A mixed-initiative mission planning system for the multi-aircraft domain," in *Proceeding of the International Conference on Automated Planning and Scheduling*. Citeseer, 2011, pp. 74–82.

[4] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence journal*, vol. 2, no. 3-4, pp. 189–208, 1971.

[5] I. Georgievski and M. Aiello, "HTN planning: Overview, comparison, and beyond," *Artifical Intelligence Journal*, vol. 222, pp. 124–156, 2015.

[6] E. D. Sacerdoti, "A structure for plans and behavior," DTIC Document, Tech. Rep., 1975.

[7] ——, "The nonlinear nature of plans," DTIC Document, Tech. Rep., 1975.

[8] A. Tate, *Project planning using a hierarchic non-linear planner*. Department of Artificial Intelligence, University of Edinburgh, 1976.

[9] ——, "Generating project networks," in *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1977, pp. 888–893.

[10] K. Currie and A. Tate, "O-Plan: the open planning architecture," *Artifical Intelligence Journal*, vol. 52, no. 1, pp. 49–86, 1991.

[11] A. Tate, B. Drabble, and R. Kirby, "O-Plan2: an open architecture for command, planning and control," in *Proceedings of the Intelligent Scheduling*. Citeseer, 1994.

[12] D. E. Wilkins, "Domain-independent planning representation and plan generation," *Artifical Intelligence Journal*, vol. 22, no. 3, pp. 269–301, 1984.

[13] ——, "Can AI planners solve practical problems?" *Computational intelligence Journal*, vol. 6, no. 4, pp. 232–246, 1990.

[14] K. Erol, J. A. Hendler, and D. S. Nau, "UMCP: A sound and complete procedure for hierarchical task-network planning." in *Proceedings of the Artificial Intelligence Planning Systems*, vol. 94, 1994, pp. 249–254.

[15] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila, "SHOP: Simple hierarchical ordered planner," in *Proceedings of the international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 968–973.

[16] D. S. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An htn planning system," *Journal of Artificial Intelligence Research*, vol. 20, pp. 379–404, 2003.

[17] M. de la Asunción, L. Castillo, J. Fdez-Olivares, Ó. García-Pérez, A. González, and F. Palao, "SIADEX: An interactive knowledge-based planner for decision support in forest fire fighting," *Artificial Intelligence Communications*, vol. 18, no. 4, pp. 257–268, 2005.

[18] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *JAIR*, pp. 253–302, 2001.

[19] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[20] J. Seipp, S. Sievers, and F. Hutter, "Fast downward cedalion," *International Planning Competition Planning and Learning Part: planner abstracts*, 2014.

[21] P. H. H. Geffner and P. Haslum, "Admissible heuristics for optimal planning," in *Proceedings of the International Conference of AI Planning Systems*, 2000, pp. 140–149.

[22] P. Haslum, B. Bonet, H. Geffner *et al.*, "New admissible heuristics for domain-independent planning," in *Proceedings of the AAAI Conference*, vol. 5, 2005, pp. 9–13.

[23] J. Hoffmann, J. Porteous, and L. Sebastia, "Ordered landmarks in planning," *Journal of Artificial Intelligence Research*, vol. 22, pp. 215–278, 2004.

[24] S. Richter, M. Helmert, and M. Westphal, "Landmarks revisited," in *Proceedings of the National Conference of the American Association for Artificial Intelligence*, vol. 8, 2008, pp. 975–982.

[25] M. Helmert, P. Haslum, J. Hoffmann, and R. Nissim, "Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces," *Journal of the Association for Computing Machinery*, vol. 61, no. 3, p. 16, 2014.

[26] R. Barták, M. A. Salido, and F. Rossi, "Constraint satisfaction techniques in planning and scheduling," *Journal of Intelligent Manufacturing*, vol. 21, no. 1, pp. 5–15, 2010.

[27] A. Lopez and F. Bacchus, "Generalizing graphplan by formulating planning as a CSP," in *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 3. Citeseer, 2003, pp. 954–960.

[28] H. A. Kautz, B. Selman *et al.*, "Planning as satisfiability." in *Proceedings of the European Conference on Artificial Intelligence*, vol. 92. Citeseer, 1992, pp. 359–363.

[29] J. Rintanen, "Planning as satisfiability: Heuristics," *Artificial Intelligence Journal*, vol. 193, pp. 45–86, 2012.

[30] ——, "Madagascar: Scalable planning with SAT," in *Proceedings of the International Planning Competition*, 2014.

[31] H. Kautz and B. Selman, "Unifying SAT-based and graph-based planning," in *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 99, 1999, pp. 318–325.

[32] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos, *Extending planning graphs to an ADL subset*. Springer, 1997.

[33] J. Koehler and J. Hoffmann, "Handling of inertia in a planning system," Tech. Rep., 1999.

[34] D. Pellier, "PDDL4J planning library," 2016. [Online]. Available: https://github.com/pellierd/pddl4j