Grounding of HTN Planning Domain

Abdeldjalil Ramoul, Damien Pellier, Humbert Fiorino, Sylvie Pesty

International Journal on Artificial Intelligence Tools

Grounding of HTN Planning Domain

Abdeldjalil Ramoul[1,2], Damien Pellier[1], Humbert Fiorino[1], Sylvie Pesty[1]

[1] *Univ. Grenoble Alpes, LIG, F-38000 Grenoble, France*
firstname.lastname@univ-grenoble-alpes.fr

[2] *Cloud Temple, 215 Avenue Georges Clemenceau,*
92024, Nanterre Cedex, France,
abdeldjalil.ramoul@cloud-temple.com

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Many Artificial Intelligence techniques have been developed for intelligent and autonomous systems to act and make rational decisions based on perceptions of the world state. Among these techniques, HTN (*Hierarchical Task Network*) planning is one of the most used in practice. HTN planning is based on expressive languages allowing to specify complex expert knowledge for real world domains. At the same time, many preprocessing techniques for classical planning were proposed to speed up the search. One of these technique, named *grounding*, consists in enumerating and instantiating all the possible actions from the planning problem descriptions. This technique has proven its effectiveness. Therefore, combining the expressiveness of HTN planning with the efficiency of the grounding preprocessing techniques used in classical planning is a very challenging issue. In this paper, we propose a generic algorithm to ground the domain representation for HTN planning. We show experimentally that grounding process improves the performances of state of the art HTN planners on a range of planning problems from the International Planning Competition (IPC).

Keywords: HTN planning, planning domain representations

1. Introduction

Act and make rational decisions based on perceptions of the world state is a central issue in intelligent and autonomous systems. Many Artificial Intelligence techniques have been developed to that purpose. Among these techniques, HTN (*Hierarchical Task Network*) planning is one of the most used in practice [1–3]. HTN planners are based on very expressive languages allowing to specify expert knowledge on real world domains. Unlike classical planning [4] where the goal is defined as a set of propositions to achieve, in HTN planning, the goal is expressed as a set of tasks to achieve, i.e. the *goal tasks*, and to which it is possible to associate different kind of constraints. A tuple (constraints, tasks) is called a *Task Network*. The search for a solution consists in decomposing the goal tasks into sub-tasks satisfying the constraints until a set of primitive sub-tasks is found: primitive tasks can be executed

by classical planning actions. The recursive decomposition of tasks into sub-tasks is performed by applying hierarchical planning operators named *methods*.

Many efficient classical planning algorithms like *Fast Forward* [5] or *Fast Downward* [6, 7] have been implemented. All of them perform a preprocessing step that consists in enumerating and instantiating all the possible actions from the planning problem descriptions. This step, named *grounding*, is crucial for many reasons. First of all, it reduces the number of the actions through different simplification mechanisms, and, consequently, the size of the research space. Secondly, generating the set of all the relevant actions allows to assess proposition reachability in the planning problem. This is a necessary prerequisite to compute efficient heuristics guiding the search process [5, 8–12]. Thirdly, the grounding allows to express planning problems in others formalisms such as CSP [13, 14] or SAT [15–18].

Combining the expressiveness of HTN planning with the efficiency of classical planners is a very challenging issue. We propose in this paper a grounding algorithm for HTN planners. This algorithm extends the grounding used in classical planners to the HTN planners. In section 2, we present the related works, and, in section 3, we define the HTN concepts. In section 4, we detail the instantiation and simplification mechanisms involved in the grounding. In section 6, we show that the grounding improves the performances of state of the art HTN planners on a range of planning problems from the International Planning Competition (IPC).

2. Related Work

Using hierarchical action representation in the automated planning community is a foundational idea. Already in 1975, Sacerdoti [19] proposed a planner called NOAH (Nets of Action Hierarchies) based on this idea. The system was built up on a data structure called *procedure net*. This data structure introduced for the first time the concept of tasks network and decomposition. These two concepts are today part of all the modern HTN planners. Each procedural net defines a hierarchy of nodes (a task network) where each node represents a particular action primitive or method at some level of detail (a decomposition). Nodes at each level of the hierarchy are linked to each others with precedence constraints. NOAH planning algorithm operates on the principle of the decomposition of an initial procedural net into a more detail procedural net in the order defined by the precedence constraints. At each decomposition, NOAH checks if the precedence constraints of the obtained procedural net are satisfied. The decomposition process ends when a procedural net containing only primitive actions is built. Many pioneering works followed NOAH:

- Nonline planner developed by [20, 21]. Nonline extends the approach of Sacerdoti and introduces a formalism describing domains in a hierarchic fashion.
- O-Plan planning system [22] and its successor O-Plan2 [23] extend the Nonline planner and proposed for the first time search control heuristics for HTN planning involving the use of condition typing, time and resources

constraints.

- SIPE (System for Interactive Planning and Execution) [24] and it successor SIPE-2 [25] is a complete hierarchical planning system that includes many features such as interleaving planning and execution, interactive plan development, replanning if failures occur during execution and the possibility to use conditional effects in action descriptions.
- UMCP (Universal Method Composition Planner) [26] is the first HTN algorithm whose correctness and completeness were proved.

The common characteristic of all these works relies on the nature of the search space explored to find a solution plan: a space of plans. No state is maintained during the search. At each search step, the tasks network obtained after a decomposition is a partial plan. The main advantage of this approach is to postpone decisions about the ordering of the actions to the execution whereas total-order planning produces a totally ordered sequence of actions.

In contrast, recent and modern HTN planners such as SHOP (Simple Hierarchical Ordered Planner) [27] maintain states during search process and explore a space of states. Each tasks network contains a representation of the state in addition to the tasks. A decomposition is applicable if and only if some constraints expressed as precondition of the decomposition hold in the state. This is very powerful because the preconditions allow to prune quickly unpromising decompositions. A comprehensive comparison of these different works is proposed in [28] and a complexity analysis of HTN planning is available in [26, 29]. Many works based on this approach were developed during the last decade. For instance, SHOP was extended to generate partial order plans [30], temporal plans [31] or plans integrating user preferences [32]. It was also extended to multi-agent systems [33], to on-line search [34] or to solve non-deterministic planning problems [35] by combining Binary Decision Diagram and HTN-based mechanisms in order to constrain the search of a solution plan. GoDel planning system [36] synthesizes classical and HTN planning. It defines a simple formalism that extends classical planning to include problem decomposition and a planning algorithm based on this formalism enable to combine HTN search when decompositions are available and classical forward-search otherwise.

Finally, [37] and [38] propose to use classical SAT and CSP solvers by encoding respectively HTN planning problem into satisfiability problem and Dynamic CSP problem. These two techniques are used to solve a variety of combinatorial and optimization problems.

In terms of applications, HTN is one of the most used planning techniques for real-world applications. For instance, it has been mainly used for automatic web services composition. HTN is well-suited for automatic web services composition because web services are described in a hierarchical manner like HTN planning formalism. Thus, the translation can be done automatically. The first use of HTN planning for automatic web services composition was done by [39,40]. In these works, web-services are automatically translated into HTN descriptions by using ontologies,

and SHOP2 planner is used to generate the sequence of web services that must be executed over the web to answer the query of an user. Many subsequent approaches were proposed : OWL-S Xplan [41] is a hybrid approach that combines the Graph-Plan based planning [42] and the HTN planning to provide a plan with incomplete web services or methods description decomposition, [43] combines HTN planning and CSP to schedule web services constraints, [44] proposes to associate Markov Decision Process and HTN planning to compute multiple composition plans, and then present the most appropriate to the user.

HTN was also widely used for decision support tools. Indeed, hierarchical reasoning used in HTN planner is "human-friendly". Decision support tools based on HTN planning techniques were applied to many areas such as search and rescue mission planning [45–48], in space exploration [49], or in multiple drones management [50].

Robotics is a classical field for automated planning. For instance, the HTN planner HATP [51] was specifically designed to solve robotics planning problems by interleaving planning and geometric reasoning. In [52–54], HTN planning techniques are used for robot navigation and gripper manipulations, etc.

3. HTN Planning Definitions and Concepts

This section provides the basic definitions to introduce HTN planning language and search procedure based on [55].

3.1. *Operators, methods and tasks*

The definition of an operator is the same as in classical planning.

Definition 3.1. An *operator* is a 3-tuple $o = (name(o), pre(o), eff(o))$. $name(o)$ is a syntactic expression of the form $t(u_1, \ldots, u_k)$ where t is the operator name and $u_1, \ldots, u_k$ 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 are logical propositions generated by the operator. Negated propositions are respectively labelled $pre^-(o)$ and $eff^-(o)$, $pre^+(o)$ and $eff^+(o)$ otherwise.

An *action* is a ground instance of an operator. An action a is *applicable* to a state s if $pre^+(a) \subseteq s$ and $pre^-(a) \cap s = \emptyset$. The resulting state s' of the application of a in a state s is defined as follows:

$$s' = \gamma(s, a) = (s \ - eff^-(a)) \cup eff^+(a)$$

Definition 3.2. A *method* is a 3-tuple $m = (name(m), pre(m), subtasks(m))$. $name(m)$ is a syntactic expression of the form $t(u_1, \ldots, u_k)$ where t is the name of the method and $u_1, \ldots, u_k$ are its parameters. $pre(m)$ is a logical expression that defines the preconditions that must be verified to apply the method, and $subtasks(m)$ is a list of subtasks of m.

A *decomposition* is a ground instance of a method. A decomposition d is applicable in a state s if $pre^+(d) \subseteq s$ and $pre^-(d) \cap s = \emptyset$.

Definition 3.3. A *task* is a syntactic expression of the form $t(u_1, \ldots, u_k)$ where t is the task name and $u_1, \ldots, u_k$ its parameters. A task is ground if all its parameters are ground. If t is an operator symbol then the task is *primitive*; otherwise, the task is *non-primitive*.

An action a accomplishes a ground primitive task (respectively a ground non-primitive task) t in a state s if $name(a) = t$ and a is applicable in s. Similarly, a decomposition d accomplishes a ground non-primitive task t in a state s if $name(d) = t$ and d is applicable in s.

Example 3.1. (Rover domain)

To illustrate these definitions, consider the *rover* domain from the International Planning Competitions. This domain deals with Mars exploration and rovers that have to cross several waypoints to collect rocks or soil samples, photograph targets and transmit data to a lander. Every rover is designed for a certain kind of ground and is not able to cross all the waypoints. Therefore the rovers have to cooperate to fulfil all the missions.

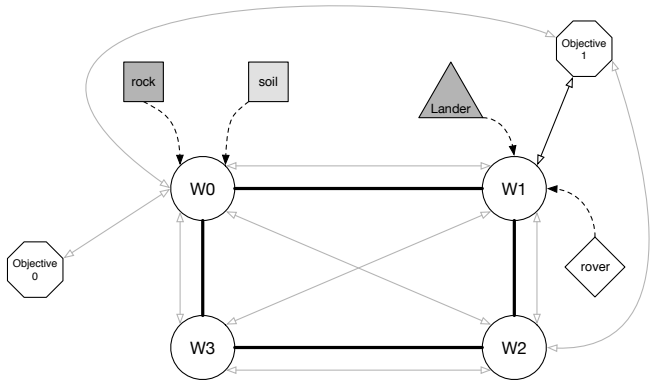


Fig. 1. Rover problem diagram.

Fig. 1 depicts a simple problem with four connected waypoints w_0, w_1, w_2, w_3: rovers can move from w_0 to w_1, from w_1 to w_2 etc. A *rover* is at w_1, a *lander* is at w_1, *rock* and *soil sample* are at w_0. The double arrows indicate visibility constraints between waypoints and objectives: $(visible\ w_0\ w_2)$, $(visible\_from\ objective_0\ w_0)$ etc. The goal is to communicate soil and rock data from W_0, to get an image of $objective_1$ and communicate it to the lander. The goal tasks are expressed as : $(get\_soil\_data\ w_0)$ $(get\_rock\_data\ w_0)$ $(get\_image\_data\ objective_1\ low\_res)$.

The operator *navigate* in the Planning Description Definition Language (PDDL) [56] is as follows:

```
(:action navigate
 :parameters (?x - rover ?from - waypoint ?to - waypoint)
 :precondition (and (available ?x) (can_traverse ?x ?from ?to)
                    (at ?x ?from) (visible ?from ?to))
 :effect (and (not (at ?x ?from)) (at ?x ?to)))
```

The operator has three parameters: $?x$ is of type "rover", $?from$ and $?to$ are of type "waypoint". *navigate* defines rover movement from one waypoint to another. *navigate* preconditions are that the rover $?x$ is available at $?from$, $?to$ is visible from $?from$, and $?x$ can move between $?from$ and $?to$. The effects are "positive" like *(at ?x ?to)* or "negative" (i.e. negated) like *(not (at ?x ?from))*.

A method *do_navigate* can be defined in PDDL style as follows:

```
(:method do_navigate
 :parameters (?x - rover ?from ?to - waypoint)
 :precondition (and (not (can_traverse ?x ?from ?to))
                    (not (visited ?mid))
                    (can_traverse ?x ?from ?mid))
 :subtasks ((navigate ?x ?from ?mid) (visit ?mid)
            (do_navigate ?x ?mid ?to) (unvisited ?mid)))
```

The method has three parameters: the parameter $?x$ is of the type "rover" and $?from$ and $?to$ are of the type "waypoint". The method is applicable if (1) the rover $?x$ is not able to cross directly from waypoint $?from$ to waypoint $?to$ (2) it has not visited the waypoint $?mid$ and finally (3) it must be able to cross from the waypoint $?from$ to $?mid$. The execution of the method involves to execute recursively the sequence of tasks: *(navigate ?x ?from ?mid)*, *(visit ?mid)*, *(do_navigate ?x ?mid ?to)* and *(unvisited ?mid))*. Note that unlike the operators, the methods can use parameters that are not declared (for instance, $?mid$). The type of these parameters have to be inferred.

3.2. *Problems and solutions*

In this section, we introduce HTN problems and solutions.

Definition 3.4. A HTN planning problem is a 4-tuple $P = (s_0, T, O, M)$ where s_0 is the initial state defined by a set of logical propositions characterizing the world, T is ordered list of initial tasks defining the goal, O is a set of operators defining the actions that can be performed, and M is a set of methods defining the possible decomposition of a task.

We now define what it means for a plan, i.e. a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ to be a solution for a planning problem $P = (s_0, T, O, M)$. In other words, what it means to accomplish the task T. Intuitively, it means that there is a

decomposition of T into π such than π is executable from s_0 and each decomposition is applicable in the appropriate state of the world. The recursive formal definition has tree cases.

Definition 3.5. Let $P = (s_0, T, O, M)$ be a HTN planning problem. The cases in which a plan $\pi = \langle a_1, \ldots, a_n \rangle$ is a *solution* for P are:

Case 1. T is an empty set of tasks. Then the empty plan $\pi = \langle\rangle$ is the solution.

Case 2. The first task $t \in T$ is primitive. Then π is a solution for P if there is an action a obtained by grounding an operator $o \in O$ such that (1) a accomplishes t, (2) a is applicable in s_0 and (3) $\pi = \langle a_2, \ldots, a_n \rangle$ is a solution plan for the HTN planning problem:

$$P' = (\gamma(s_0, a_1), T - \{t\}, O, M)$$

Case 3. The first task $t \in T$ is non-primitive. Then π is solution if there is a decomposition d obtained by grounding a method $m \in M$ such that s accomplishes t and π is solution for the HTN planning problem

$$P' = (s_0, T - \{t\} \cup \mathit{subtasks}(d), O, M)$$

3.3. *HTN planning procedure*

Algorithm 1 shows the HTN generic and non-deterministic procedure for solving a HTN planning problem. The procedure is based directly on the recursive definition of a solution plan for HTN planning problem.

The generic HTN procedure takes as input a problem $P = (s_0, T, O, M)$ where s_0 is the initial state, $T = \langle t_1, t_2, ..., t_k \rangle$ is a list of tasks, O, the set of operators, M, the set of methods. First, the procedure tests if the list of tasks T is empty (line 2). In this case, no task have to be executed thus the empty plan is returned. Then the procedure get the first task t_1 of the list T. Two cases must be considered depending on the type of t_1:

Case 1. If t_1 is primitive (line 3) then the procedure computes the set of all the ground actions that accomplishes t_1 and that are applicable in s_0 (line 4). If there is no action (line 5), the procedure fails because no action accomplishes the goal task t_1. Then the procedure non-deterministically chooses an action that accomplishes the task (line 6) and calls it-self recursively on the planning problem $P' = (\gamma(s_0, a_1), T - \{t_1\}, O, M)$ (line 7). Finally, if the recursive call to the procedure fails to find a plan π, it returns failure (line 8); otherwise it returns the plan that is the concatenation of a and π (line 9).

Case 2. If t_1 is non-primitive (line 10) then the procedure computes the set of ground decompositions that accomplishes t_1 and that are applicable in s_0 (line 11). If there is no decomposition to accomplishes t_1 (line 12) then the procedure returns failure. Then the procedure non-deterministically

Algorithm 1: HTN(s_0, T, O, M)

1 Let $T = \langle t_1, \ldots, t_k \rangle$
2 **if** $k = 0$ **then** return the empty plan $\langle\rangle$
3 **if** t_1 *is primitive* **then**
4 $\quad A \leftarrow \{a \mid a$ is an action obtained by grounding an operator $o \in O$ such that a accomplishes t_1 and a is applicable in $s_0\}$
5 $\quad$ **if** $A = \emptyset$ **then** return failure
6 $\quad$ non-deterministically choose an action $a \in A$
7 $\quad \pi \leftarrow HTN(\gamma(s_0, a), \langle t_2, \ldots, t_k \rangle, O, M)$
8 $\quad$ **if** $\pi =$ ***failure*** **then** return **failure**
9 $\quad$ **else** return $a \oplus \pi$
10 **else if** t_1 *is a non-primitive task* **then**
11 $\quad D \leftarrow \{d \mid d$ is a decomposition obtained by grounding a method $m \in M$ such that d accomplishes t_1 and d is applicable in $s_0\}$
12 $\quad$ **if** $D = \emptyset$ **then** return failure
13 $\quad$ non-deterministically choose a decomposition $d \in D$
14 $\quad$ return HTN$(s_0, \mathit{subtasks}(d) \oplus \langle t_2, \ldots, t_k \rangle, O, M)$

chooses a decomposition d that accomplishes the task t_1 (line 13) and recursively returns the solution plan for the problem $P' = (s_0, \mathit{subtasks}(d) \oplus \langle t_2, \ldots, t_k \rangle, O, M)$ (line 14).

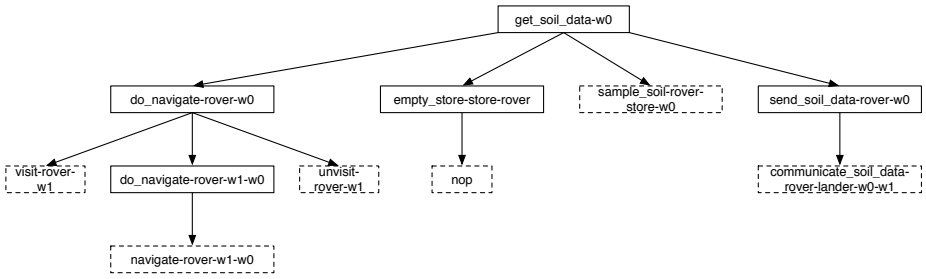


Fig. 2. Example of total order task decomposition: primitive and non-primitive task are respectively depicted with dotted and plain lines

To illustrate the recursive decomposition of the HTN procedure, Fig. 2 shows the decompositions tree of the task *get_soil_data-w0* from the rover domain into primitive tasks. At level 1, the *get_soil_data-w0* task is decomposed into three non-primitive tasks *do_navigate-rover-w0*, *empty_store-store-rover* and *send_soil_data-rover-w0* and the primitive task *sample_soil-rover-store-w0*. One possible solution plan for the task *get_soil_data-w0* is as follows: ⟨*visit-rover-w1*, *navigate-rover-*

w1-w0, *unvisit-rover-w1*, *sample_soil-rover-store-w0*, *communicate_soil_data-rover-lander-w0-w1*⟩.

4. The Problem of Grounding HTN Domain

The grounding process generates all the possible instances of the operators and the methods from the objects (constants) defined in a planning problem. To do so, all the typed parameters in the operators and the methods are replaced by constants of the same type. The number of instances depends on the number of parameters in the operators and methods as well as in the number of constants in each type. Let x_i be a parameter and $D(x_i)$ be its "domain" i.e. the set of all the possible constants that can be associated with this parameter ($|D(x_i)|$ is the size of $D(x_i)$). The number of instances $I(o)$ of an operator o with n parameters $\{x_1, ..., x_n\}$ having as domain $D(x_i)$ is:

$$I(o) = \prod_{i=1}^{n} |D(x_i)|$$

It is obvious that the number of instances rapidly increases with the number of parameters. For instance, the number of actions generated by the grounding of the operator *communicate_soil_data (?x - rover ?l - lander ?p1 - waypoint ?p2 - waypoint ?p3 - waypoint)* in a rover problem of IPC-5 is 14 million = $(14 \times rover) \times (1 \times lander) \times (100 \times waypoint1) \times (100 \times waypoint2) \times (100 \times waypoint3)$.

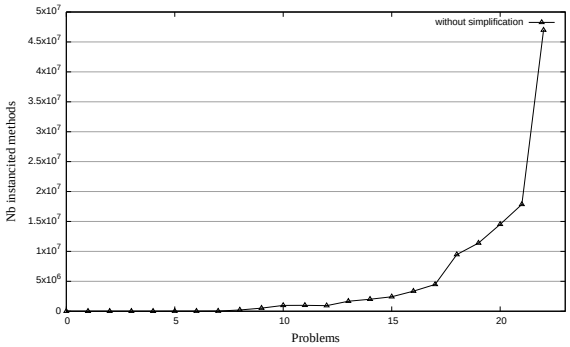


Fig. 3. Number of ground methods without simplification.

In addition to generate all operator instantiations, the problem of grounding for HTN domains entails to generate all the possible method instantiations. Because primitive tasks are necessarily defined in methods, the parameter number in methods is higher than in operators. As a consequence, method grounding generates much more instances than operator grounding. For instance, the number of ground methods of 23 *rover* problems with increasing size (with respect to the number of

objects) from the IPC-5 are shown in Fig.3: we see that the number of ground methods increases rapidly with the number of objects in the problems. There are 14 objects in problem 0, 68 in problem 11 and 158 in problem 22. There are 2 times more objects in the problem 22 than in the problem 11, when in the same time there is 48 times more ground methods. Therefore, efficient simplification mechanisms are essential as we know that the number of methods strongly influences the branching factor of the HTN search algorithm.

5. The Grounding Algorithm for HTN Domain

Our grounding process is outlined in Fig. 4. Starting from a HTN planning problem described by using an extended version of PDDL to HTN, we rely on the process presented in [57] to produce a table of *inertias* as detailed in section 5.1. Based on this inertia table, the operator grounding and simplification (see section 5.2) produces a semi-instantiated problem with actions. Finally, the hierarchical problem is totally instantiated and simplified (section 5.3).

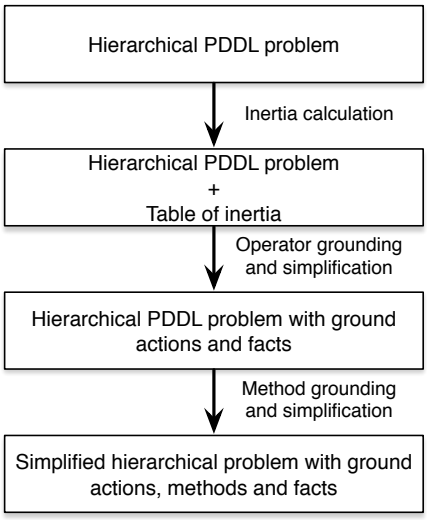


Fig. 4. Overview of the hierarchical problems grounding algorithm.

5.1. *Inertia calculation*

The concept of *inertia* defined in [57] represents facts (propositions in action preconditions or effects) that are never produced or consumed by any operator. There are two types of inertia:

- **Positive inertia** are facts that are never produced by the positive effects

of any operator. If a positive inertia is not in the initial state of the problem, it will never be true in any state of the problem.
- **Negative inertia** are facts that are never consumed by the negative effects of any operator. If a negative inertia is in the initial state of the problem, it will always be true in any state of the problem.

If a fact is both a positive and a negative inertia, it is an *inertia*. A *fluent* is a fact that is neither positive nor negative inertia and may appear or disappear from one state to another. In practice, the inertia calculation is done in two steps during the operator grounding process as explained in [57]. The first one is done before the grounding, with partially ground propositions to handle the grounding complexity and reduce the operators and the methods during the process. The second one called ground inertia is performed after the grounding to completely simplify ground operators and methods.

Table 1. Inertia of the Rover domain.

Proposition name	Positive inertia	Negative inertia	Status
available	No	Yes	Negative inertia
at	No	No	Fluent
visible	Yes	Yes	Inertia
can_traverse	Yes	Yes	Inertia
communicated_soil_data	No	Yes	Negative inertia
communicated_rock_data	No	Yes	Negative inertia
at_soil_sample	Yes	No	Positive inertia
at_rock_sample	Yes	No	Positive inertia
at_lander	Yes	Yes	Inertia
visited	No	No	Fluent

Table 1 shows examples of inertias in the rover domain. The facts *can_traverse*, *visible* and *at_lander* do not appear in positive and negative effects of any operator making them positive and negative inertias. The positive inertias *at_soil_sample* and *at_rock_sample* do not appear in positive effects of the operators: they will never be consumed. *at* and *visited* are fluents so they can be added or removed during a state transition.

5.2. *Operator grounding and simplification*

The operator grounding process generates the relevant operator instances. It follows the four steps algorithms as shown in Fig. 5.

5.2.1. *Normalization of operator logical expressions*

For each logical expression defined in the preconditions and the effects of an operator containing implications or quantifiers, a reformulation into a Conjunctive or Disjunctive Normal Form is a prerequisite. We apply the following logical transformations:

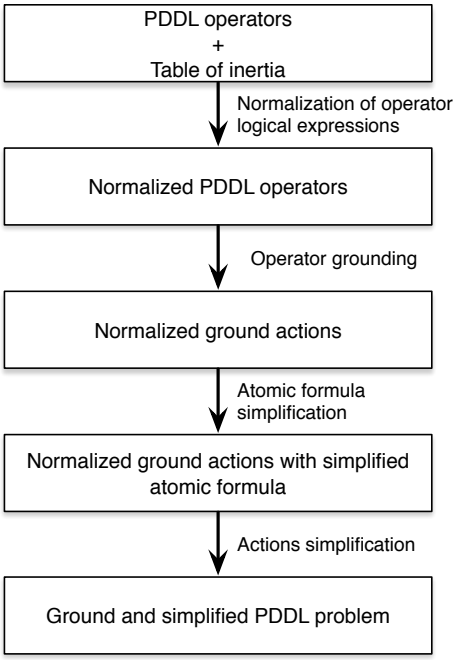


Fig. 5. Overview of the operator grounding and simplification algorithm.

- $\phi \rightarrow \varphi \Rightarrow \neg\phi \wedge \varphi$
- $\neg(\phi \wedge \varphi) \Rightarrow \neg\phi \vee \neg\varphi$
- $\forall(?x - type) \Rightarrow x_1 \wedge x_2 \wedge ... \wedge x_n$
- $\exists(?x - type) \Rightarrow x_1 \vee x_2 \vee ... \vee x_n$

5.2.2. *Operator grounding*

The operator grounding consists in generating all its possible instances. For instance, the action *navigate* of the following example is a possible instance of the operator *navigate*:

```
(:action navigate
 :parameters (rover w1 w0)
 :precondition (and (available rover) (can_traverse rover w1 w0)
                    (visible w1 w0) (at rover w1))
 :effect (and (not(at rover w1)) (at rover w0)))
```

The parameters are grounded as follows : $?x \rightarrow rover$, $?from \rightarrow w1$, $?to \rightarrow w0$. The grounding process loops on the operators set, and defines the matching objects for each parameter with respect to their type. If the problem is not typed, it is necessary to perform a type inference step before the grounding. This is detailed in [57].

5.2.3. *Atomic formula simplification*

The atomic formula simplification has to be done as early as possible. Using the table of inertia calculated before, the atomic formula simplification consists in evaluating the preconditions and effects formula contained in the operators to $true$ or $false$. Let a proposition p contained in an atomic formula of an operator and s_0 the initial state of a planning problem, the simplification of the atomic formulas follows these rules:

- **If** p is a positive inertia and $p \notin s_0$ **then** p is simplified to **false**.
- **If** p is a negative inertia and $p \in s_0$ **then** p is simplified to **true**.
- **Else** p cannot be simplified.

For instance, considering the inertia table of the rover domain, and assuming that the proposition *(can_traverse rover w1 w0)* is *true* in the initial state, then the simplification will replace it by *true*. Now, assume that *(can_traverse rover w1 w0)* is *false* in the initial state, then the simplification will replaces it by *false* and a formula in the precondition or the effect of an operator such as

```
(and (available rover) (can_traverse rover w1 w0)
     (visible w1 w0) (at rover w1))
```

can be simplified to *false*.

5.2.4. *Actions simplification*

The goal is to find and remove actions that will never be applied because of an atomic formula simplification. As atomic formulas can be simplified to *true* or *false*, preconditions and effects can be simplified by applying logical transformation rules:

- 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 will never be applied; if the effect is *false*, the application of the action produces an inconsistent state.
- If all the action effects are *true*, the action can be removed from the problem because it does not produce any change.

5.3. *Method grounding and simplification*

Method grounding and simplification follows the five stages given in Fig. 6: (1) the preconditions of the methods are normalized; (2) the type of the undeclared parameters used in the preconditions are inferred; (3) the methods are instantiated; (4) the preconditions of the method are simplified based on the atomic formula simplification previously presented and (5) irrelevant methods, i.e., methods with tasks whose actions have been deleted in the previous simplification or with preconditions simplified to *false*, are deleted from the problem.

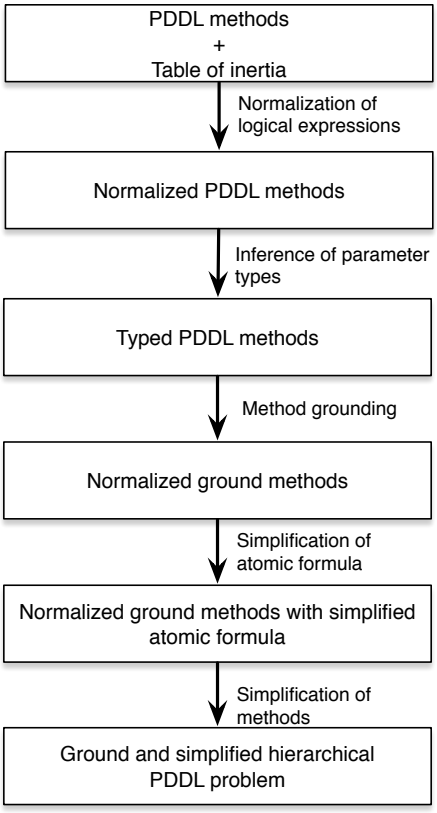


Fig. 6. Overview of the method grounding and simplification algorithm.

5.3.1. *Normalization of logical expressions*

The normalization of a method transforms its preconditions into Conjunctive Normal Form (see section 5.2.1 for more details)

5.3.2. *Inference of parameter types*

In methods, parameters can be used in subtasks and precondition declaration without being declared in the method parameters. Therefore, these parameters have no explicit declared types. Their type must be inferred before performing method grounding. The type inference process has two steps:

Step 1. Inferring types from subtasks:

- Get the list of subtasks T containing undeclared parameters.
- For each subtask $t \in T$, get the operators o and the methods m that accomplish t.

- For each subtask $t \in T$, get the declared types in the parameters of o or m. If there are two types A and B, where B is a subtype of A, keep type B.
- If several types with no inheritance link are found, then an error is reported.

Step 2. Inferring types from preconditions:

- Get the set P of propositions used in the preconditions of the methods.
- For each proposition $p \in P$, infer possible types from the set of typed predicate list defined in the domain.
- If several types with no inheritance link are obtained, then an error is reported. Otherwise, keep the most general type.

5.3.3. *Method grounding*

The grounding of a method consists in replacing all its parameters by constants. Each combination of constants associated to the method parameters produces a ground method. For instance, a ground method of *do_navifage* is as follows:

```
(:method do_navigate
 :parameters (rover w1 w0)
 :precondition (and (not (can_traverse rover w1 w0))
                    (not (visited w3))
                    (can_traverse rover w1 w3))
 :subtasks ((navigate rover w1 w3) (visit w3)
            (do_navigate rover w3 w0) (unvisit w3)))
```

The ground method *do_navigate-rover-w1-w0* previously introduced is an instance of the method *do_navigate* with the following associations: $?x \rightarrow rover$, $?from \rightarrow w1$, $?to \rightarrow w0$, $?mid \rightarrow w3$ ($?mid$ inferred type is *waypoint*, and it is associated to the constant $w3$).

5.3.4. *Simplification of atomic formulas*

This simplification aims at evaluating the atomic formulas contained in the method preconditions to *true* or *false* from the inertias. The simplification is based on the same procedure presented for the operators.

5.3.5. *Simplification of methods*

The method simplification aims at identifying and deleting the methods containing preconditions that will never be verified in the planning problem. Two kinds of simplification are realized on the methods:

Precondition based simplification. It relies on the evaluation of the logical expressions, which can be simplified as *true* or *false* in the previous step. The simplification is done on the following rules:

- If the precondition is simplified as *true*, the precondition is removed from the method, as it is always verified.
- If the precondition is simplified as *false*, the method is deleted: this precondition will never be verified in the planning problem, and this method will never produce a solution plan.

Task based simplification. It aims at deleting methods containing primitive tasks that cannot be applied. Assuming that the operator simplification is made before the method simplification, it is performed as follows:

(1) Get the set T of primitive tasks in the method to be simplified,
(2) For every task $t \in T$, check if the relevant action for t was deleted during the operator simplification step. If this is the case, then the method is deleted.

6. Evaluation

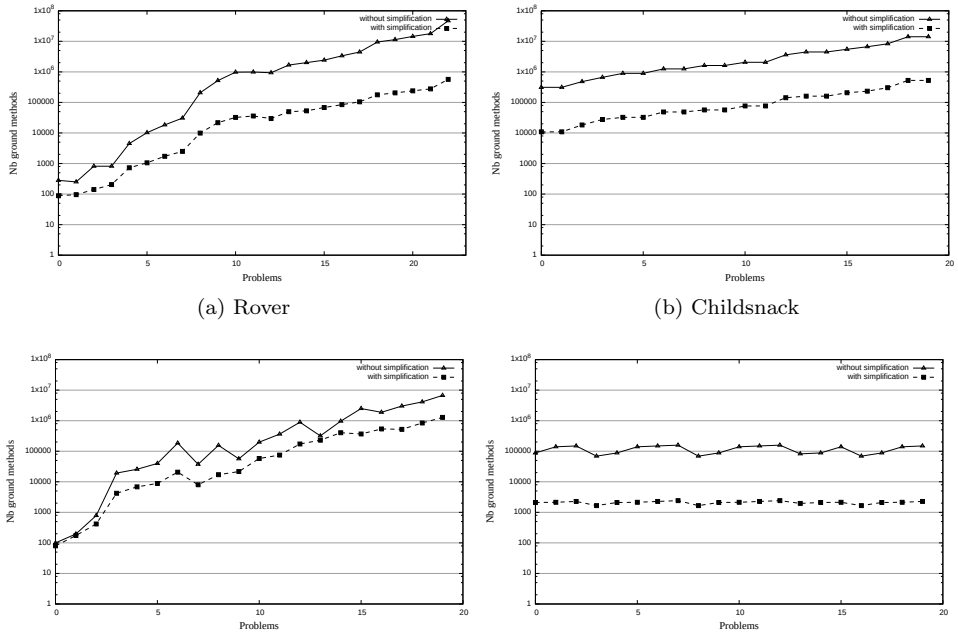
In this section, we investigate to what extent the grounding improves the performance of HTN algorithms. We have coded the *Ground Total Order Hierarchical Planner "G_TOHP"* as implementation of the algorithm described in section 3.3, and we have compared its performances in terms of processing time and plan lengths with a classical HTN planner. G_TOHP is in JAVA. It is based on the PDDL4J planning library [58]. This library includes lexical and syntactical analysis modules for classical PDDL planning. We have added the lexical and syntactical analysis of methods, and we have implemented the HTN grounding algorithm.

6.1. *Evaluation of the simplification rate*

The first evaluation criteria focuses on the number of ground methods generated with or without the simplification process. The simplification rate allows to see how much the HTN problems are simplified in terms of ground methods and actions. To that end, we have improved the set of tests presented in [59] by evaluating the simplification algorithm on four HTN versions of the International Planning Competition domains: rover, childsnack, satellite and barman. Each domain contains at least twenty problems of growing size.

Fig. 7 shows the number of ground methods resulting from the full method grounding process and from the grounding process without simplification. The simplification rate is the method count without simplification on the simplified methods. Fig. 7 shows that the number of ground methods decreases sharply after simplification specifically in problems with large sizes. We see in Fig. 7(a, b and c) that the difference in the number of ground methods increases exponentially with the increase in problem sizes. In Fig. 7(a) for instance, the simplification rate is 30.16 in problem 10 with 997,202 unsimplified methods on 32400 simplified methods, and 82.58 in the problem 22 with 46,980,192 unsimplified methods on 568,850 after simplification. In Fig. 7(d), the simplification rate is high, and remains stable. It ranges

between 41.65 and 65.57 because of the stable size of the barman problems that contain between 37 and 43 objects.



(a) Rover (b) Childsnack

(c) Satellite (d) Barman

Fig. 7. Logarithmic representation of the number of ground methods before and after simplification on four planning domains: Rover, Satellite, Childsnack, Barman from IPC (International Planning Competition)

6.2. *Evaluation of the planning performance with ground problems*

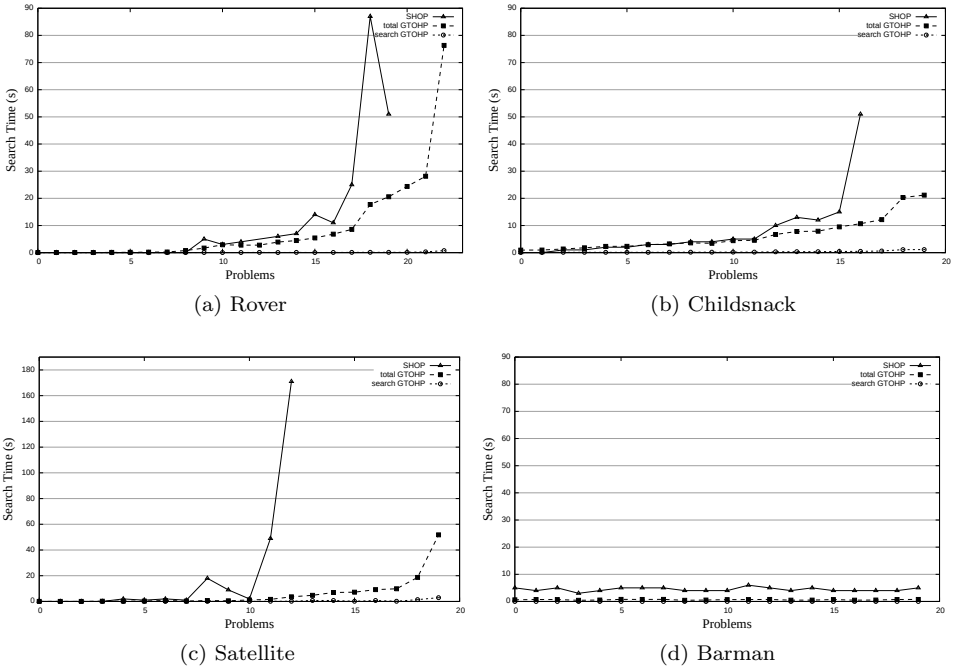
6.2.1. *Experimental framework*

These experiments compare our implementation of *SHOP* classical HTN planner [27] with *G_TOHP* and *FastDownard* [6] on four planning domains: Rover, Childsnack and Satellite and Barman. We show that *G_TOHP* outperforms the other planners.

All the results were obtained with a multi-core Intel Core i7 clocked at 2.2 GHZ and 16GB DDR3 RAM. The algorithm evaluations follow the International Planning Competition *agile* track criteria [60]: a score is calculated for each planner based on its own plan search time compared with other competing algorithms best time. For G_TOHP, the search time results are presented in the form of two graphs, the first one represents the total processing time which is equal to the grounding and simplification time plus the search time, and the second presents the search time

separately in order to show the part taken by the search in the total process. It should also be noted that the grounding and simplification time is easily deducible since it represents the difference between the total time and the search time. In addition to the search time, we also compare the plan lengths obtained with each algorithm knowing that, especially in HTN, plan lengths are strongly related to domain definitions.

All the search time results are presented in Fig. 8. The X-axis represents the planning problems, and the Y-axis represents the processing time (in seconds) to find a solution plan. The processing time results are shown in the same decimal scale from 0 to 90 except for figure (c) Which has a scale from 0 to 180 due to an isolated point with a value equal to 171 seconds. A maximum search time is set to 10 minutes and no displayed result for a problem means that the planner was not able to find a solution in the allocated time. Fig. 9 shows the lengths of solution plans, with the number of actions displayed on the Y-axis and the planning problems on X-axis.



(a) Rover (b) Childsnack

(c) Satellite (d) Barman

Fig. 8. Time comparison between SHOP and G_TOHP on the planning domains : Rover, Childsnack, Satellite and Barman

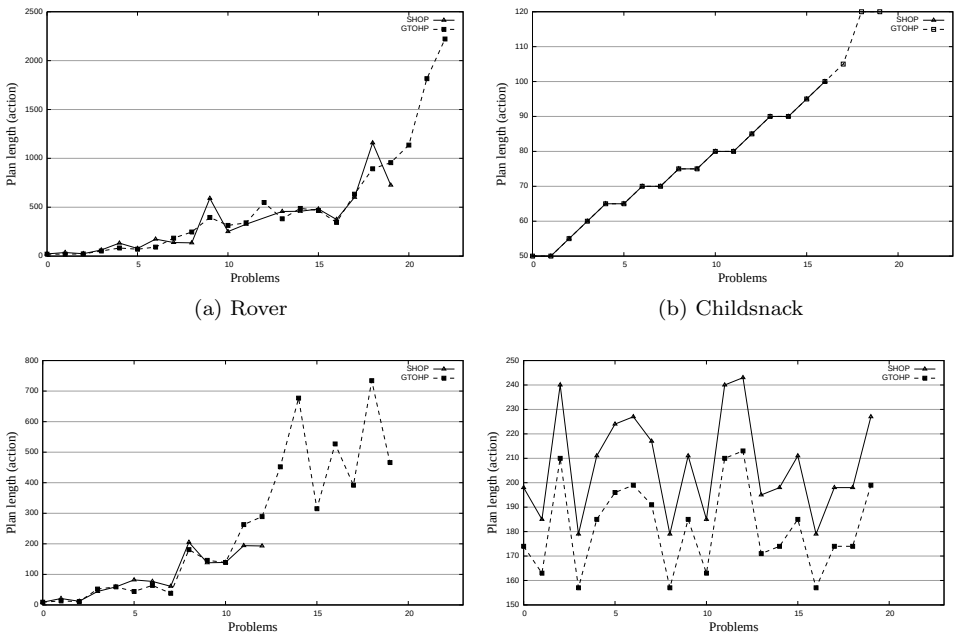
6.2.2. *Results analysis*

In the four domains, G_TOHP always takes less time to find the solution than SHOP. This shows that the grounding makes the search faster. In addition, in all the tested problems, the search time represents less than 5% of G_TOHP's total processing time, and it never exceeds 3 seconds, even in the largest problems. This is interesting in re-planning situations when goals change. Due to the fact that grounding and simplification process is only affected by changes in the initial state, the operators or the methods, no more grounding process is needed in replanning situations involving only goals changes. However, in case of changes in the initial state, operators or methods, a complete grounding and simplification process is necessary in order to guarantee the integrity of the ground problem. In Fig. 8(a, b and c), the difference in search time between the two algorithms is very small for small problems, and grows rapidly with the problem size. In Barman domain, the difference in search time is stable around 4 seconds. This is due to the stability of the Barman problem size. SHOP cannot find a solution in the allocated time from problem 19 in the rover domain, problem 16 in childsnack and problem 12 in satellite, whereas G_TOHP finds solutions for all the problems in the allocated time (it takes 76.33 sec to find a plan for the largest rover problem with 7844 decompositions, 21.19 sec for the largest childsnack problem with 25 decompositions, and 51.77 sec for the largest satellite problem with 64550 decompositions).

Table 2. Scores of the algorithms Fast Downward, G_TOHP and SHOP

	Rover			Childsnack			Satellite			Barman		
Pb	FD	GTOHP	SHOP	FD	GTOHP	SHOP	FD	GTOHP	SHOP	FD	GTOHP	SHOP
00	1	1	1	0	1	1	1	1	1	0	1	0.51
01	1	1	1	0.49	1	1	1	1	1	0	1	0.55
02	1	1	1	0.34	1	0.99	1	1	1	0.38	1	0.52
03	1	1	1	1	0.62	0.61	1	1	1	0	1	0.51
04	1	1	1	1	0.66	0.66	1	1	0.33	0	1	0.51
05	1	1	1	1	0.71	0.73	1	1	0.39	0	1	0.53
06	1	1	1	0	1	0.96	1	1	0.37	0	1	0.54
07	1	1	1	0	1	0.99	1	1	0.38	0	1	0.53
08	1	1	1	0	1	0.91	1	1	0.33	0	1	0.51
09	1	0.92	0.63	1	0.89	0.84	1	1	0.42	0	1	0.51
10	1	0.86	0.79	0	1	0.94	0.84	1	0.75	0	1	0.54
11	1	0.98	0.81	0	1	0.95	1	0.96	0.40	0	1	0.51
12	1	0.72	0.63	0	1	0.83	0.71	1	0.38	0	1	0.53
13	1	0.76	0.64	0	1	0.81	0.53	1	0	0.27	1	0.51
14	1	0.79	0.59	0	1	0.84	0.64	1	0	0	1	0.50
15	1	0.74	0.62	0	1	0.83	0.42	1	0	0	1	0.54
16	1	0.74	0.54	0	1	0.59	0	1	0	0	1	0.52
17	1	0.70	0.47	0	1	0	1	0.66	0	0	1	0.52
18	1	0.70	0.54	0	1	0	0.66	1	0	0	1	0.56
19	1	0.68	0	0	1	0	1	0.60	0	0	1	0.54
20	1	0.70	0	-	-	-	-	-	-	-	-	-
21	1	0.70	0	-	-	-	-	-	-	-	-	-
	22	19.00	15.32	4.84	18.89	14.51	16.81	19.22	7.75	0.65	20	10.51

In Table 2. on the Rover domain, *Fast Downward* obtains a perfect score of 22/22. This means that Fast Downward was the faster to find a plan for all the problems. G_TOHP is second with a score of 19/22 and SHOP is third with a score of 15.35: G_TOHP is 13.63% less efficient than Fast Downward, and 16.72% more efficient than SHOP. G_TOHP is the most efficient on the Childsnack domain with a score of 18.89/20 and was respectively 63.85% and 19.88% more efficient than Fast Downward and SHOP. The good scores obtained by HTN planners comparing to Fast Downward show the effectiveness of HTN planning on problems requiring numerous backtracking: the decomposition methods allow to sharply restrict the search space. On satellite domain, G_TOHP obtains a score of 19.22/20 and is 10.98% more efficient than Fast Downward with a score of 16.8, and is 52.14% more efficient than SHOP with a score of 7.75. On Barman domain, G_TOHP obtains 20/20 with all plans found in less than one second. SHOP is half less efficient with a score of 10.51. Fast Downward obtains a very bad score of 0.65, as it finds only two plans on 20. The overall score obtained by Fast Downward is 44.30/82, 77.12/82 by G_TOHP and 48.10/82 by SHOP: G_TOHP is 52.93% more efficient than Fast Downward and 46.80% more efficient than SHOP. All these results confirm that HTN planning using grounding is more efficient in terms of execution time than classical SHOP with a clear advantage on large problems.



(a) Rover (b) Childsnack

(c) Satellite (d) Barman

Fig. 9. Plan lengths comparison between G_TOHP and SHOP on the planning domains : Rover, Childsnack, Satellite and Barman

In Fig. 9 (a, b, c and d) are presented the lengths of plans produced by G_TOHP and SHOP during the experiment. The two algorithms generate plans of almost the same length. The plan length increases with the problem size except in Barman domain where it remains stable, because in this domain, the number of goal tasks does not change a lot from one problem to another. Unlike G_TOHP, SHOP does not find plans for all the problems (excepted for Barman). In the Childsnack domain, the two algorithms find exactly the same plans. Knowing that SHOP takes more time than G_TOHP to find these plans, this shows that even by following the same decompositions, G_TOHP is more efficient than SHOP since it searches in a smaller search space.

7. Conclusion

We have presented in this paper a grounding approach for HTN planning. It reuses the grounding and simplification techniques used in classical planning, and proposes new rules to instantiate HTN methods. We have demonstrated the effectiveness of our approach on different planning domains, which obtains much shorter search times than a classical HTN approach.

The future developments and extensions of this work will focus on the following issues:

- Develop search heuristics for HTN planning: having all the possible actions and decompositions makes possible to assess task reachability for HTN planning problem. Task reachability is a necessary step to compute efficient heuristics guiding the search process.
- Develop new SAT and CSP encodings for HTN planning to exploit the efficiency of SAT and CSP solvers. These two technologies are very successful in solving combinatorial and optimization problems. However, they are underused specifically for HTN planning. Having all the possible actions and decompositions allows to investigate new efficient CSP and/or SAT encodings.

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* 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* **2**(3-4) (1971) 189–208.
5. J. Hoffmann and B. Nebel, The FF planning system: Fast plan generation through heuristic search, *Journal of Artificial Intelligence Research* (2001) 253–302.

6. M. Helmert, The Fast Downward planning system, *Journal of Artificial Intelligence Research* **26** (2006) 191–246.
7. J. Seipp, S. Sievers and F. Hutter, Fast Downward cedalion, *International Planning Competition Planning and Learning Part: planner abstracts* (2014).
8. H. Geffner and P. Haslum, Admissible heuristics for optimal planning, in *Proceedings of the International Conference of AI Planning Systems* 2000, pp. 140–149.
9. P. Haslum, B. Bonet and H. Geffner, New admissible heuristics for domain-independent planning, in *Proceedings of the Association for the Advancement of Artificial Intelligence Conference* **5** 2005, pp. 9–13.
10. J. Hoffmann, J. Porteous and L. Sebastia, Ordered landmarks in planning, *Journal of Artificial Intelligence Research* **22** (2004) 215–278.
11. S. Richter, M. Helmert and M. Westphal, Landmarks revisited, in *Proceedings of the National Conference of the American Association for Artificial Intelligence* **8** 2008, pp. 975–982.
12. 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* **61**(3) (2014) p. 16.
13. R. Barták, M. A. Salido and F. Rossi, Constraint satisfaction techniques in planning and scheduling, *Journal of Intelligent Manufacturing* **21**(1) (2010) 5–15.
14. A. Lopez and F. Bacchus, Generalizing graphplan by formulating planning as a CSP, in *Proceedings of the International Joint Conference on Artificial Intelligence* **3** 2003, pp. 954–960.
15. H. A. Kautz and B. Selman, Planning as satisfiability, in *Proceedings of the European Conference on Artificial Intelligence* **92** 1992, pp. 359–363.
16. J. Rintanen, Planning as satisfiability: Heuristics, *Artificial Intelligence Journal* **193** (2012) 45–86.
17. J. Rintanen, Madagascar: Scalable planning with SAT, in *Proceedings of the International Planning Competition* 2014.
18. H. Kautz and B. Selman, Unifying SAT-based and graph-based planning, in *Proceedings of the International Joint Conference on Artificial Intelligence* **99** 1999, pp. 318–325.
19. E. Sacerdoti, The nonlinear nature of plans, in *Proceedings of the International Joint Conference on Artificial Intelligence* 1975, pp. 206–214.
20. A. Tate, *Project planning using a hierarchic non-linear planner* (Department of Artificial Intelligence, University of Edinburgh, 1976).
21. A. Tate, Generating project networks, in *Proceedings of the International Joint Conference on Artificial Intelligence* Morgan Kaufmann Publishers Inc. 1977, pp. 888–893.
22. K. Currie and A. Tate, O-Plan: the open planning architecture, *Artifical Intelligence Journal* **52**(1) (1991) 49–86.
23. A. Tate, B. Drabble and R. Kirby, O-Plan2: an open architecture for command, planning and control, in *Proceedings of the Intelligent Scheduling* 1994.
24. D. E. Wilkins, Domain-independent planning representation and plan generation, *Artifical Intelligence Journal* **22**(3) (1984) 269–301.
25. D. E. Wilkins, Can AI planners solve practical problems?, *Computational intelligence Journal* **6**(4) (1990) 232–246.
26. K. Erol, J. A. Hendler and D. Nau, UMCP: A sound and complete procedure for hierarchical task-network planning., in *Proceedings of the Artificial Intelligence Planning Systems* **94** 1994, pp. 249–254.
27. 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.
28. I. Georgievski and M. Aiello, HTN planning: Overview, comparison, and beyond, *Artifical Intelligence Journal* **222** (2015) 124–156.
29. R. Alford, P. Bercher and D. W. Aha, Tight bounds for HTN planning with task insertion, in *Proceedings of the International Joint Conference on Artificial Intelligence* 2015, pp. 1502–1508.
30. D. S. Nau, T. Au, O. Ilghami, U. Kuter, J. M. Murdock, D. Wu and F. Yaman, SHOP2: An htn planning system, *Journal of Artificial Intelligence Research* **20** (2003) 379–404.
31. 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* **18**(4) (2005) 257–268.
32. S. Sohrabi, J. A. Baier and S. A. McIlraith, HTN planning with preferences, in *Proceedings of the International Joint Conference on Artificial Intelligence* 2009, pp. 1790–1797.
33. D. Pellier and H. Fiorino, A unified framework based on HTN and POP approaches for multi-agent planning, in *Proceedings of the International Conference on Intelligent Agent Technology* 2007, pp. 285–288.
34. B. Marthi, S. J. Russell and J. Wolfe, Angelic hierarchical planning: Optimal and online algorithms, in *Proceedings of the International Conference on Automated Planning and Scheduling* 2008, pp. 222–231.
35. U. Kuter, D. S. Nau, M. Pistore and P. Traverso, Task decomposition on abstract states, for planning under nondeterminism, *Artif. Intell.* **173**(5-6) (2009) 669–695.
36. V. Shivashankar, R. Alford, U. Kuter and D. S. Nau, The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning, in *Proceedings of the International Joint Conference on Artificial Intelligence* 2013, pp. 2380–2386.
37. A. D. M. and S. K., Encoding HTN planning in propositional logic, in *Proceedings of the International Conference on Artificial Intelligence Planning Systems* 1998, pp. 190–198.
38. S. Pavel and B. Roman, Encoding HTN planning as a dynamic CSP, in *Proceedings of the International Conference on Principles and Practice of Constraint Programming* 2005, p. 868.
39. E. Sirin, B. Parsia, D. Wu, J. Hendler and D. Nau, HTN planning for web service composition using SHOP2, *Web Semantics: Science, Services and Agents on the World Wide Web* **1**(4) (2004) 377–396.
40. D. Wu, B. Parsia, E. Sirin, J. Hendler and D. Nau, Automating DAML-S web services composition using SHOP2, in *International Semantic Web Conference* Springer 2003, pp. 195–210.
41. M. Klusch, A. Gerber and M. Schmidt, Semantic web service composition planning with owls-xplan, in *Proceedings of the Association for the Advancement of Artificial Intelligence Fall Symposium on Agents and the Semantic Web* 2005, pp. 55–62.
42. A. Blum and M. Furst, Fast Planning Through Planning Graph Analysis, *Artificial Intelligence* **90**(1-2) (1997) 281–300.
43. I. Paik and D. Maruyama, Automatic web services composition using combining HTN and CSP, in *Proceedings of the International Conference on Computer and Information Technology* IEEE 2007, pp. 206–211.
44. K. Chen, J. Xu and S. Reiff-Marganiec, Markov-htn planning approach to enhance flexibility of automatic web service composition, in *Proceedings of the International Conference on Web Services* IEEE 2009, pp. 9–16.
45. K. L. Myers, W. M. Tyson, M. J. Wolverton, P. A. Jarvis, T. J. Lee and M. desJardins, Passat: A user-centric planning framework, in *Proceedings of the International NASA*

Workshop on Planning and Scheduling for Space 2002, pp. 1–10.
46. K. L. Myers, P. Jarvis, M. Tyson and M. Wolverton, A mixed-initiative framework for robust plan sketching., in *Proceedings of the International Conference on Automated Planning and Scheduling* 2003, pp. 256–266.
47. J. Allen and G. Ferguson, Human-machine collaborative planning, in *Proceedings of the International NASA Workshop on Planning and Scheduling for Space* 2002, pp. 27–29.
48. D. W. Aha, L. A. Breslow and H. Muñoz-Avila, Conversational case-based reasoning, *Applied Intelligence* **14**(1) (2001) 9–32.
49. T. Kichkaylo, C. van Buskirk, S. Singh, H. Neema, M. Orosz and R. Neches, Mixed-initiative planning for space exploration missions, in *Proceedings of the International Conference on Automated Planning and Scheduling Workshop on Moving Planning and Scheduling Systems into the Real World* 2007.
50. G. Bevacqua, J. Cacace, A. Finzi and V. Lippiello, Mixed-initiative planning and execution for multiple drones in search and rescue missions, in *Proceedings of the International Conference on Automated Planning and Scheduling* 2015.
51. R. Lallement, L. De Silva and R. Alami, HATP: An HTN planner for robotics, in *In proceedings of the International Conference on Automated Planning and Scheduling Workshop on Planning and Robotics* 2014.
52. R. Hartanto, Fusing dl reasoning with HTN planning, *KI-Künstliche Intelligenz* **25**(1) (2011) 81–84.
53. T. Belker, M. Hammel and J. Hertzberg, Learning to optimize mobile robot navigation based on HTN plans, in *Proceedings of the International Conference on Robotics and Automation* **3**, IEEE 2003, pp. 4136–4141.
54. M. Weser, D. Off and J. Zhang, HTN robot planning in partially observable dynamic environments, in *Proceedings of the International Conference on Robotics and Automation* IEEE 2010, pp. 1505–1510.
55. M. Ghallab, D. Nau and P. Traverso, *Automated planning: theory & practice* (Elsevier, 2004).
56. M. Ghallab, A. Howe, G. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld and D. Wilkins, *PDDL: The Planning Domain Definition Language*. Artificial Intelligence Planning Systems, (1998).
57. J. Koehler and J. Hoffmann, Handling of inertia in a planning system, tech. rep. (1999).
58. D. Pellier, PDDL4J planning library (2016), https://github.com/pellierd/pddl4j.
59. A. Ramoul, D. Pellier, H. Fiorino and S. Pesty, HTN planning approach using fully instantiated problems, in *Proceeding of the International Conference on Tools in Artificial Intelligence* 2016, pp. 113–120.
60. L. L. Carlos, J. C. Sergio and H. Malte, Automating the evaluation of planning systems, *Artificial Intelligence Commun.* **26**(4) (2013) 331–354.