Symbolic and Geometric Planning for teams of Robots and Humans

Raphaël Lallement

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée par :

Présentée et soutenue le *08 Septembre 2016* **par :**

RAPHAËL LALLEMENT

Symbolic and Geometric Planning for teams of Robots and Humans

---

JURY

MALIK GHALLAB	Directeur de recherche, LAAS	Président du Jury
FRANÇOIS CHARPILLET	Directeur de recherche, INRIA	Rapporteur
ALLESSANDRO SAFFIOTTI	Professeur, University of Örebro	Rapporteur
RACHID ALAMI	Directeur de recherche, LAAS	Directeur de thèse
IVAN MAZZA	Maître de conférence, University of Seville	Examinateur

---

École doctorale et spécialité :

MITT : Domaine STIC : Intelligence Artificielle

Unité de Recherche :

Laboratoire d'Analyse et d'Architecture des Systèmes

Directeur(s) de Thèse :

Rachid Alami et *Malik Ghallab*

Rapporteurs :

Alessandro Saffiotti et *François Charpillet*

Abstract

Hierarchical Task Network (HTN) planning is a popular approach to build task plans to control intelligent systems. It is used in a variety of fields such as robotics, mission control, and so on. Its popularity comes from the intuitive representation that it uses, furthermore the planner uses that representation to cut down the search space (compared to classical planning approaches). In robotics, the domain representation consists in specifying high-level instructions on how robots and agents should perform tasks, while also giving the planner enough flexibility to choose the lower-level steps and their ordering.

This thesis presents the HATP (Hierarchical Agent-based Task Planner) planning framework which extends the traditional HTN planning domain representation and semantics by making them more suitable for roboticists, and by offering human-awareness capabilities. This planning framework can be seen as a planning laboratory thanks to the ease to develop new domains and the possibility to extend its capabilities.

When computing human-aware robot plans, it appears that the problems are very complex and highly intricate. To deal with this complexity we have integrated a geometric planner to reason about the actual impact of actions on the environment and allow to take into account the affordances (reachability, visibility). To benefit from this geometric layer, the symbolic layer refines the actions in the current plan it is computing by assessing their geometric feasibility in the current world state. In addition the geometric layer computes and sends back symbolic literals representing the effective outcome on the environment of the particular instance of actions chosen by the geometric planner. The symbolic layer can use these literals to reason about the side effects of the actions which allows to partially tackle the ramification problem. This thesis presents in detail this integration between two heterogeneous planning layers and explores how they can be combined to solve new classes of robotic planning problems.

One of the main issues we face when combining the two layers is the planning time that grows rapidly and prevents any significantly-big problem to be solved in reasonable time. We study and propose in this thesis an improvement to the backtrack mechanism to drive the symbolic search toward more promising solutions using a combination of symbolic and geometric information.

Version française

La planification HTN (Hierarchical Task Network, ou Réseau Hiérarchique de Tâches) est une approche très souvent utilisée pour produire des séquences de tâches servant à contrôler des systèmes intelligents. Cette méthode est employée dans différents domaines parmi lesquels la robotique, le contrôle de missions, etc. La reconnaissance dont elle bénéficie vient de la façon intuitive qu'elle a de représenter les problèmes. De plus elle adopte cette description pour accélérer la recherche (par rapport à la planification dite classique). Encoder un domaine robotique consiste à spécifier des instructions de haut-niveau décrivant comment les robots et les agents doivent accomplir leurs tâches tout en assurant assez de flexibilité au planificateur pour lui permettre de choisir l'ordre des étapes de bas-niveau.

Cette thèse présente le planificateur HATP (Hierarchical Agent-base Task Planner, ou Planificateur Hiérarchique centré Agent) qui étend la planification HTN classique en enrichissant la représentation des domaines et leur sémantique afin d'être plus adaptées à la robotique, tout en offrant aussi une prise en compte des humains. HATP peut être vu comme un laboratoire de planification de par sa capacité à être amélioré et la facilité à ajouter de nouveaux domaines.

Quand on souhaite générer un plan pour des robots tout en prenant en compte les humains, il apparaît que les problèmes sont complexes et fortement interdépendants. Afin de faire face à cette complexité, nous avons intégré à HATP un planificateur géométrique apte à déduire l'effet réel des actions sur l'environnement et ainsi permettre de considérer la visibilité et l'accessibilité des éléments. Pour exploiter cette couche géométrique, la couche symbolique affine les actions du plan lors du calcul en en vérifiant la faisabilité dans l'état courant du monde. En outre la couche géométrique produit et renvoie des littéraux décrivant le résultat effectif d'une action particulière (instanciée) sur l'environnement quand celle-ci a été choisie par le planificateur géométrique. La couche symbolique peut alors faire face à une partie du problème de la ramification en s'appuyant sur ces littéraux qui incluent notamment les effets de bords des actions. Cette thèse se concentre sur l'intégration de ces deux planificateurs de nature différente et étudie comment par leur combinaison ils permettent de résoudre de nouvelles classes de problèmes de planification pour la robotique.

Une des difficultés majeures de cette combinaison de couches vient du temps de calcul qui croît trop rapidement empêchant de fait la résolution de problèmes conséquents en un temps raisonnable. Nous étudions puis proposons un certain nombre d'améliorations au mécanisme dit de backtrack (ce qui signifie de revenir sur les choix faits plus tôt dans le calcul d'un plan) afin de mieux diriger la recherche vers des solutions plus prometteuses, le tout en utilisant des informations aussi bien symboliques que géométriques.

REMERCIEMENTS

Je souhaite commencer en remerciant mes directeurs de thèse: Rachid Alami pour ses conseils, ses idées, nos discussions et son temps tout au long de ma thèse et pour m'avoir offert l'opportunité de faire une thèse au sein de l'équipe RIS. Mais aussi Malik Ghallab, bien que désigné tardivement comme co-directeur, il a su m'aider notamment à formaliser mes idées.

Je dois aussi remercier les différents partenaires du projet européen ARCAS dans lequel s'inscrit ma thèse, car c'était un projet très stimulant. En particulier Juan Cortes, Alexandre Boeuf, Ivan Mazza et Carmelo Jesus Fernández Agüera Tortosa avec qui j'ai eu l'occasion de collaborer le plus souvent, mais aussi Anibal Ollero pour avoir diriger ce projet.

Merci aux deux rapporteurs, Alessandro Saffiotti et François Charpillet, ainsi qu'aux membres de mon jury de soutenance d'avoir consacré une partie de leur temps à la relecture de mon manuscrit ainsi que pour l'attention porté à ma soutenance, aboutissant à des questions et discussions intéressantes.

Merci à tous ceux qui m'ont donné envie de faire de la recherche, d'abord Claudio Zito et Rustam Stolkin en me permettant de faire un stage à l'Université de Birmingham, ensuite tous les enseignants de la formation IUP-SI notamment Viviane Cadenat pour avoir répondu à mes nombreuses questions, mais aussi Simon Lacroix en m'introduisant au LAAS lors d'un stage et enfin les quelques doctorants et amis m'ayant convaincu de franchir le pas de et de m'engager dans une thèse, notamment Cordélia Robinson.

Un merci tout particulier à Mamoun Gharbi pour l'intégralité du travail que nous avons réalisé ensemble durant nos thèse respectives mais très interdépendantes. Mais aussi à Lavindra de Silva pour son aide et ses conseils m'ayant aidé tel un mentor. Enfin tous les collègues avec qui j'ai eu l'occasion de travailler, Sandra Devin, Grégoire Milliez et Michelangelo Fiore ainsi que tous les autres collègues du LAAS ayants rendu ce séjour très agréable et permettant des discussions très riches.

Enfin je remercie ma famille et et mes amis de m'avoir soutenu tout au long de cette difficile période qu'est la thèse. Je me suis toujours senti très entouré me permettant de toujours faire face aux moments difficiles mais aussi de pouvoir célébrer les succès.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

In order for robots to be able to cooperate with humans we need them to exhibit intelligent behaviours. To reason about the near future is a stepping stone to reach such intelligence. This reasoning process is called planning and has been divided in smaller problems: plan the next actions with the task planning, plan the motions the robots have to execute with motion planning, and many other forms of planning.

The goal of this thesis is to study task planning in robotic domain and some challenging problems for planners that are the cost-based search for plan quality essentially in the context of multi-agent and more particularly human-robot collaboration problems. We also we claim to contribute to the challenging problem of combining in a single plan step symbolic and geometric reasoning. Among the many approaches we have chosen Hierarchical Task Network (HTN) planning. This method relies on a description of the problem based on a set of tasks using a hierarchy which resembles the way the humans reason about problems by decomposing them into sub-problems depending on the context. In addition to being easy to understand and design, those task networks are also very efficient for task planning. Indeed they allow to find a solution plan in a time short enough for many real-world applications. This combination of expressiveness and planning efficiency makes HTN a "good" robotics planning system.

However one of the limitations of symbolic planning, when it is applied to robots, is the inability to take into account the geometry: certain actions a robot decides to carry out can be infeasible due to the current state of the environment. For instance if the robot decides to move an object to a given table, if this table is covered with other objects it is then impossible to execute the move action. Furthermore there is no practical way to handle this kind of problem with HTN, partial solutions can be found but the planner does not reason on the actual geometry so the solutions will be very limited and inefficient.

In this thesis we formulate a way to integrate a geometric task planner with the HTN planner in order to tackle this limitation. The planner provides the system with a motion planner but also offers a number of geometric reasoning capabilities: it can extract symbolic literals, it maintains the geometric plan, it allows to better estimate the cost of actions. We also propose a set of improvements to this integration to tighten the link between the two planners. The structure of the thesis is presented in the next section.

1.1 Structure of the thesis

We start in Chapter 2 by presenting a brief overview of task planning, and use it to motivate our decision to use HTN planning. Then we define more precisely the HTN formalism used in the rest of this thesis. This formalism not only introduces a new graphical representation of the HTN planning algorithm but also identifies the different type of backtracks involved in the search of a solution.

Chapter 3 is focused on our planner HATP, and present it as it was before the integration with the geometric task planner. We present its differences to classical HTN, and first comes the domain representation: we use state-variables, a custom domain representation and a language designed to help computer scientists. When listing those differences we also prove that its custom domain representation is equivalent to the classical one and that we do not loose any property. We highlight some differences on the search algorithm; it is indeed cost driven in order to be both faster than the classical one and find better solutions. Additionally it offers multi-agent capabilities allowing to split the solution plan into several streams of actions corresponding to the different agents. Since it was first designed to address Human-Robot interactions, we show the features it has to improve the plans to be more socially acceptable. This chapter ends with some insights on HATP implementation and integration in a robotic architecture. Appendix A gives more details and also presents some improvements done solely on the implementation.

We introduce the integration between symbolic planning and geometric planning in Chapter 4. We begin by defining a formalism to have a common ground when discussing the state of the art. It is followed by a thorough study of related work, sorted by categories of approaches and summarised in a condensed table. We proceed with our formalism and algorithm, presenting the Geometric Task Planner and its integration with HATP. We give details on the internal representation of the geometric actions at symbolic level, and how this level exploits literals extracted from the geometric world state. We then show that those literals can be used to face the ramification problem allowing our complete planning system to address new classes of problems. We use several examples to illustrate our system. We finally present some enhancements done in order to reduce the combinatorial growth due to the integration of the two planners. The first is a system of constraints given by the symbolic layer to reduce the geometric search space. The second proposes a way to use the geometry as a “common-sense” heuristic in the domain modelling: the domain experts can test future geometric states to decide which tasks decomposition to use. Finally we introduce and use the notion of high-level goals

attached to tasks in order to allow the planner to reason about the side-effects of actions and tasks to trigger a backtrack if the decomposition does not reach its objectives.

Chapter 5 starts with an analysis of one of the main issues of this integration: the very long planning time. So the chapter depicts a method that improves the backtrack mechanism to speed-up the search. We present a mechanism that uses criteria to evaluate the likeliness of the different backtrack points to lead to a successful solution. We then propose some criteria corresponding to recurring problems in our domains: criteria to cope with a problem occurring when placing an object, or when creating a stack. We then propose an idea for a heuristic search in the HTN tree, instead of depth first, that would benefit both from symbolic and geometric knowledge. Since this idea is just a proposal, only its formalism and algorithm are shown.

The last chapter, Chapter 6, showcases some domains where HATP is being used. The first is the ARCAS project where a fleet of aerial robots endowed with a manipulator arm have to assemble buildings. In this context an assembly planner has been integrated with our symbolic-geometric planner such that the assembly plan to build the structure becomes an input of our system. Hence we can compute at the same time the task allocation and assess the feasibility of the actions. We then present a domain where a robot is asked to explain its plan to a co-worker while adapting to his/her level of knowledge. The principle is to verbalise and monitor only the part relevant to the user knowledge preventing any useless speech and allowing for more flexible execution if the user is already knowledgeable of the tasks to accomplish.

We finish with a conclusion and some perspectives for future works.

1.2 Contributions

In the scope of this thesis we started by creating a clean and formalised way to integrate a geometric task planner with symbolic planning to allow planning for robotics problem. Indeed planning both at symbolic and geometric levels allows to reason about the feasibility of the actions and their effective outcome on the world state. The Geometric Task Planner we are using allows us to instantiate the actions in the symbolic plan in the geometric world and test they are feasible. GTP stores a plan which is the geometric counterpart of the symbolic plan in order to retain the new world states. Additionally this allows GTP to propose alternative placement and trajectories for the actions already in the plan should the symbolic plan be invalid.

We have studied the state of the art and have proposed an organisation of the references into several groups according to their planning-level-combination strategy. Thanks to this survey, we hope that people from the community of Combined Task And Motion Planning (CTAMP) will be able to better understand how the different publications relate to each other.

We have formalised, categorised and illustrated the different types of backtracks from the decomposition process, then extended it to also cover the symbolic-geometric combination. We then worked on procedures to exploit this knowledge and sort them so we could have a faster planner. We think

that this formalisation could be used by other algorithms, using forward-chaining or hierarchical planning, for their own formalisation.

We have then created a way to retrieve the effect of an action on the geometry to exploit this knowledge at the symbolic level. Thus we can reason about the side effects of actions, allowing us to address the ramification problem. In order to achieve this integration we extended an existing HTN planner, HATP developed at LAAS, but also created a new abstract layer over a motion planner, called GTP.

We then have introduced a mean to encode common sense strategies in this planning system: we can virtually test an action to choose the decomposition to use, we also integrated high-level goals to identify the decompositions that fail to achieve their objectives. The strategy to virtually test an action would be beneficial to any planner that uses hierarchy and in the CTAMP community. While the high-level goals could be used by any hierarchical planner in order to identify wrong decompositions during the planning process.

We have created a way to classify the backtrack points in order to determine the most promising one when we encounter an error while planning. We created several criteria each focusing on identifying a specific error and selecting the corresponding backtrack points. Although this work proved to be working, it needs a bit more maturity before being applicable to other planners, however the principle and the classification mechanism could already be integrated in other planners whether from CTAMP or purely symbolic planning.

It opened a prospect to a heuristic search in HTN that should allow to have a more directed search toward more promising and more efficient solutions.

We have developed a wide variety of domains to use with HATP in context such as but not limited to: building assembly using a fleet of aerial manipulators, verbalisation and negotiation of plans with a human, table-top manipulation, examples from the classical planning community.

We have dedicated some effort to the implementation to make it more stable, and to document it to allow an easier distribution of the source code. Thanks to this integration work, HATP is usable by the robotics community, moreover it is distributed under open-source license.

1.3 Publications associated to this thesis

Lallement, R., de Silva, L., and Alami, R. (2014). **HATP: An HTN Planner for Robotics**. In *ICAPS Workshop on Planning and Robotics*, pages 20–27. The content corresponds mainly to Chapter 3 to present HATP. The last part of the paper mentions the integration with the geometric planner GTP, which is the focus of Chapter 4. This paper mostly aimed at presenting HATP to the robotics community, hence was focused on highlighting how the planner has been adapted to the robotic problems.

Alami, R., Gharbi, M., Vadant, B., Lallement, R., and Suarez, A. (2014). **On Human-aware Task and Motion Planning Abilities for a Teammate Robot**. In *RSS Workshop Human-Robot Collaboration for*

Industrial Manufacturing. This paper aims at presenting a complete robotics architecture. HATP is not presented in details, however its integration into the LAAS architecture is presented specifically in an industrial human-aware context. Section 3.4 shows this architecture.

de Silva, L., Lallement, R., and Alami, R. (2015). **The HATP Hierarchical Planner: Formalisation and an Initial Study of its Usability and Practicality**. In *International Conference on Intelligent Robots and Systems*, pages 6465–6472. This paper does a in-depth study of HATP language to compare it to the wide-spread SHOP language. A part of the paper is a benchmark of HATP implementation against JSHOP in order to prove that even though HATP is not really optimised it still allows real-world applications. We more specifically helped at designing and encoding the benchmark and documenting the details about the implementation. This thesis presents an extension to the study on the language in Appendix Section A.2.1.

Gharbi, M., Lallement, R., and Alami, R. (2015). **Combining Symbolic and Geometric Planning to Synthesize Human-Aware Plans: Toward More Efficient Combined Search**. In *International Conference on Intelligent Robots and Systems*. This paper presents our contributions on improving the combined search, which is depicted in Section 4.5, we introduce enhancements to extract knowledge from the geometric layer to use it into the symbolic layer (and vice-versa). We carried out the improvements on the symbolic layer and helped define the formalisation since it is inspired by the one used in the task planning community.

Milliez, G., Lallement, R., Fiore, M., and Alami, R. (2015). **Using Human Knowledge Awareness to Adapt Collaborative Plan Generation, Explanation and Monitoring**. In *International Conference on Human-Robot Interaction*. This paper illustrates the use of HATP in the context of human interactions, more precisely how to verbalise an HTN such that the human can understand such that he/she learns from a robot. The content is presented in Section 6.2. In this work, we helped implementing the tools that manipulate the plan structure and implemented all the symbolic planning capabilities responsible for accounting the human level of knowledge.

CHAPTER 2

TASK PLANNING BACKGROUND

This first chapter is a brief introduction to task planning, however it does not aim at providing yet another extensive description of task planning, it is a prelude to our problem. Indeed there are already numerous publications that focus on describing this domain, the most notable one is the book Ghallab et al. (2004). From the broad question "Why is task planning necessary ?", we go down to "Why HTN is more suited to our problem ?". Hence this chapter is not really meant to be a survey of the state of the art but just an introduction. However later in this thesis we need a proper survey of the works related to the symbolic-geometric planning hence Section 4.3 presents it.

2.1 Why is task planning needed and how does it work ?

This section aims at highlighting the need for task planning in our main study cases: Human-Robot Interactions and multi-robot cooperation. Due to the relative complexity raised by the variety of problems we need a mechanism capable of handling such issues. Moreover interacting with a human implies that the robotic system must exhibit certain features and behaviours (socially acceptable) to ensure the success of the objectives set.

2.1.1 Purpose of task planning

The book (Ghallab et al., 2004, Chap 1.1, p.1) starts with the following introduction: "Planning is the reasoning side of acting. It is an abstract, explicit deliberation process that chooses and organizes actions by anticipating on their expected outcome. This deliberation aims at achieving as best as possible some prestated objectives. Automated planning is an area of Artificial Intelligence (AI) that studies this deliberation process computationally.". Indeed what opposes automated planning to

other approaches is the reasoning about the actions and their organisation in time to achieve the given goals.

It is possible to control a robotic system using a *behavioral* mechanism, Brooks (1987) and Arkin (1998) argue that planning is not necessary to let a robot act. The principle is to create tasks to achieve according to high-level goals. And using a behavioural system the robot can act in order to achieve these goals. The problem with such mechanism is the foresight, the robots decides only the next action to take and does not plan ahead. It allows the robot to address certain specific goals whereas we want our robots to achieve more complex goals. On the other hand a planner, using a decision process, can answer such complex goals.

2.1.2 Different symbolic planning approaches

Over the past few decades the Artificial Intelligence (AI) field has been very active producing an abundant number of approaches to automated planning. Many of which are based on a mechanism that builds a *solution plan* by reasoning about the actions available and their different possible sequencing. There are many different ways to formalise such mechanism, the most obvious –and historically the first– is the state-space planning.

In a state-space planner such as STRIPS Fikes and Nilsson (1971), the search uses an exhaustive representation of the complete states in order to plan. Besides, this planner works using a backward search: the process starts with the goal and tries to go toward the initial state. This backward chaining like the similar approach of forward chaining benefited of extension such as heuristic search, one notable example being FF planner Hoffmann and Nebel (2001).

Another common approach is the plan-space planning, which paternity is attributed to Sacerdoti and its NOAH planner Sacerdoti (1974). The principle is to iteratively build a plan without relying on an explicit representation of the current state. A solution found by plan-space planning is represented as a set of actions and causal links, hence it is more flexible allowing a better control during execution.

A notable approach is the planning and acting community that tries to solve both the symbolic planning and the execution of this plan as it is built. This approach is the topic of the book Ghallab et al. (2016) and is growing in popularity. Although we prefer to use an offline method to increase our confidence in the plan produced, we still take into account a part of the real world thanks to our connection to the geometric planner as we demonstrate it in this manuscript.

The last method we present here is actually the one used in the rest of the thesis, the rest of this chapter presents it in more details. It was first proposed by Tate (1977) and is fundamentally different in that it does not plan actions to fulfil a given goal, it performs a set of *tasks*. To do so it uses a network of tasks, with tasks representing different levels of abstraction hence creating a hierarchy, therefore it is called Hierarchical Task Network (HTN) planning.

2.1.3 A first glimpse at Hierarchical Task Networks

The next section presents the Hierarchical-Task-Network formalism that is used in the rest of the thesis. This section is focused on some notable examples of HTN planning and some more recent variants based on hierarchical planning.

The main advantage of hierarchical planning comes from the network of tasks provided by the *domain expert* that designed it. This serves as a heuristic to the planner, it is a domain-dependent knowledge that drives the (domain-independent) planner toward the goal. The main limitation is the lack of guarantee for completeness, if the task network does not represent all possible cases then the planner is incomplete and may not find the best solution plan. HTN planning has been more used in planning applications than any other planning methods, for instance it has been used in: robotics Morisset and Ghallab (2002), video games Kelly et al. (2007), and many other fields.

The main motivation behind using hierarchical planning in robotics comes from its expressiveness and efficiency. It is easy to understand the tasks and their hierarchy: it helps developing new problems and can even be understood by people outside the planning community which is really valuable in robotics where many fields of expertise must cooperate. While being expressive it still is fast enough to be used in real-world applications.

The most famous HTN planner is surely SHOP2, by Nau et al. (2003) (first introduced in Nau et al. (2001)). It is the successor of SHOP, and is known for its efficiency[1].

Some variants appeared based on task networks with hierarchy as well, GoDeL by Shivashankar et al. (2013). This planner uses the task hierarchy as much as possible but as soon as it can not find a solution it tries to infer subgoals (based on landmarks techniques) and if it is still not enough it falls back to classical planning, namely action chaining.

2.2 Hierarchical Task Networks, planning with knowledge

This section presents the formalism for HTN planning, without being extensive it serves as the base for the rest of the thesis. The first section presents the actual formalism while the second section proposes a novel graphical illustration of the algorithm, mainly focused on the different types of choices HTN has to make which creates backtrack points.

2.2.1 Formal definition of (total-order) HTN planning

The formalism introduced here is largely inspired from Ghallab et al. (2004). But first let us present some basic vocabulary:

Operator: as in classical planning, is something that can be carried out by an agent (or a set of agents).

[1] It received a "distinguished performance" award during the 2002 International Planning Competition, and was the planner with the highest rate of success: it solved 899 out of the 904 problems.

Action: is a ground instance of an operator, we use the terms operator and action interchangeably in this thesis.

Method: is a recipe for decomposing an abstract task into more primitive subtasks.

Primitive task: is primitive in the sense that it maps to operators.

Abstract task: (or non-primitive tasks) needs to be decomposed into primitive tasks to be executed, several methods may decompose it.

Task: can be either primitive or abstract as shown above.

Domain: is the finite set of tasks that the planner can use to find the solution.

Problem: is the goal to achieve expressed as a task to accomplish.

The general idea of HTN planning consists in decomposing the goal task to accomplish, using the tasks available. At each iteration the planner decomposes a non-primitive task using a method, and the process continues until there are only actions. Since our implementation of HTN planning uses total ordering we limit the formalism to this case, the next Section 3 will present the difference between this general total-order HTN and our planner HATP.

An HTN *planning problem* is the 3-tuple $\mathscr{P} = \langle d, s_0, \mathscr{D} \rangle$, where d is the goal to achieve expressed as a task to accomplish, s_0 is the initial state, and $\mathscr{D}$ is an HTN *planning domain*. An HTN planning domain is the pair $\mathscr{D} = \langle \mathscr{A}, \mathscr{M} \rangle$ where $\mathscr{A}$ is a finite set of operators and $\mathscr{M}$ is a finite set of HTN *methods*.

Formally an action is defined by the 4-tuple:

$$a = \langle \text{name}(a), \text{task}(a), \text{precond}(a), \text{effects}(a) \rangle$$

where name(a) is the name and parameters of the action, task(a) the primitive task achieved by this operator, precond(a) a set of predicates that must hold true for the action to be applicable, effects(a) the effects of the action.

A method is the 5-tuple:

$$m = \langle \text{name}(m), \text{task}(m), \text{precond}(m), \text{subtasks}(m), \text{constr}(m) \rangle$$

where name(m) corresponds to the name and parameters of the method, task(m) the abstract task (non-primitive task) this method can be applied to, precond(m) specifies when the method is applicable, subtasks(m) is the set of tasks which m uses to refine task(m), and constr(m) is the set of precedence constraints[2] between the tasks in subtasks(m).

[2]The precedence constraint: $u \prec v$ means that all subtasks of u must be done before any subtask of v can start (applicable to actions as well). It is important to note that we are not using all possible constraints from the classical definition of HTN.

The planning process works by selecting applicable methods from $\mathscr{M}$ and applying them to abstract tasks in d in a depth-first manner. At each iteration, this will typically result in decomposing a non-primitive task by selecting an applicable method (its preconditions hold true in the current state). The process continues until d has only actions left. At any stage of the planning if no applicable method can be found for an abstract task, the planner essentially "backtracks" and tries an alternative method for an abstract task refined earlier. Also at any stage of the planning process we can compute the current symbolic state s by applying the effects of all the actions in the decomposition d (while enforcing their precedence constraints).

In more detail, d is a task network: $d = \langle U, C \rangle$ where U is a set of task nodes and C is a set of precedence constraints. The planning process consists in decomposing a task node $u \in U$ using a method m. Such that m is an instance of a method in $\mathscr{M}$, the task$(m) = t_u$ (t_u is the task of the task node u) and each predicate in precond(m) holds in the current symbolic state s. The method m decomposes u into subtasks(m) with:

$$\delta(d, u, m) = ((U - \{u\}) \cup \text{subtasks}(m), C' \cup \text{constr}(m))$$

Note that C' is a modified copy of C from d: if subtasks$(m) = \{u_1, u_2\}$, then the constraint $u \prec v$ (v is another task) is replaced with $u_1 \prec v$ and $u_2 \prec v$.

Sometimes during the planning process several methods from $\mathscr{M}$ can realise the task t_u, then the planner creates a *backtrack point* called BP with a *backtrack alternative*[3] for each possible method $m \in \mathscr{M}$. Two labels are associated to BP: type$(BP) = decompo$ and task$(BP) = m$. The "type" function gives the type of the backtrack point (it can be: $decompo$, $bind$, $order$) and the "task" function associates a primitive or non-primitive task to BP. The planner performs a depth-first search, making choices and creating the corresponding backtrack points, if it fails it goes back to a previous backtrack point.

Furthermore it can occur that a method has subtasks with parameters that are not grounded. In this case a variable binding occurs: a backtrack point BP is created and for each possible value of the ungrounded parameters a backtrack alternative is added. With type$(BP) = bind$ and task$(BP) = m$. Actually each alternative stores m', a copy of m such that all the parameters of the subtasks are grounded.

In total-order HTN, even though the solution plan is totally ordered, the domain $\mathscr{D}$ allows for methods to have partially-ordered subtasks: some method subtasks can have only a limited set of ordering constraints (precedence constraints). To obtain all the totally-ordered possible orders we linearise the partial-order. For instance if we have: $m = \langle \ldots, \text{subtasks}(m) = \{u_1, u_2, u_3\}, \text{constr}(m) = \emptyset \rangle$, we obtain the following orders: $\{\{u_1 \prec u_2 \prec u_3\}, \{u_1 \prec u_3 \prec u_2\}, \ldots\}$, a total of six orders are possible. For each of those orders the planner will create an alternative in a backtrack point BP with type$(BP) = order$ and task$(BP) = m$. Again each alternative saves m', a copy of m with all the constraints to obtain totally-ordered subtasks.

[3] For clarity reasons, from here on we call "backtrack alternatives" or just "alternatives" the different values for a choice that is stored thanks to a backtrack point.

Thanks to this process the partial plan π' can be computed at any stage of the planning, then applying all the actions it contains (and respecting the causal links) we can compute the complete current state s. Thanks to this computation it is possible to have an evaluable predicate in any preconditions, a function that performs elaborate computation on the current state s while being considered as a literal: it must hold true (along with the other literals) for the planner to consider the preconditions valid.

This formalism should make it obvious that the domain expert provides knowledge to the planner in the form of the hierarchy of tasks. Indeed the task hierarchy allows the planner to reason only on a limited set of task, additionally the expert can provide an order for the tasks which also prevents any order which further reduces the number of attempts before finding the solution plan.

2.2.2 Illustration of the algorithm

This section illustrates some of the internal steps of the HTN algorithm using graphic strips, and it focuses only on some non-trivial cases. This allows to define a precise representation that is used later in the thesis. The norm used will be presented at the same time as the HTN algorithm is illustrated. Only some elements of the algorithm are presented but they are all focused on the same fictitious domain in order to tie together the different cases.

2.2.2.1 Method selection

The first HTN mechanism presented is the method selection, when an abstract task must be decomposed and there are several methods available. Figure 2.1 depicts both an extract of the domain and the way the decomposition tree grows (how the partial plan is changed). In the "PARTIAL PLAN"

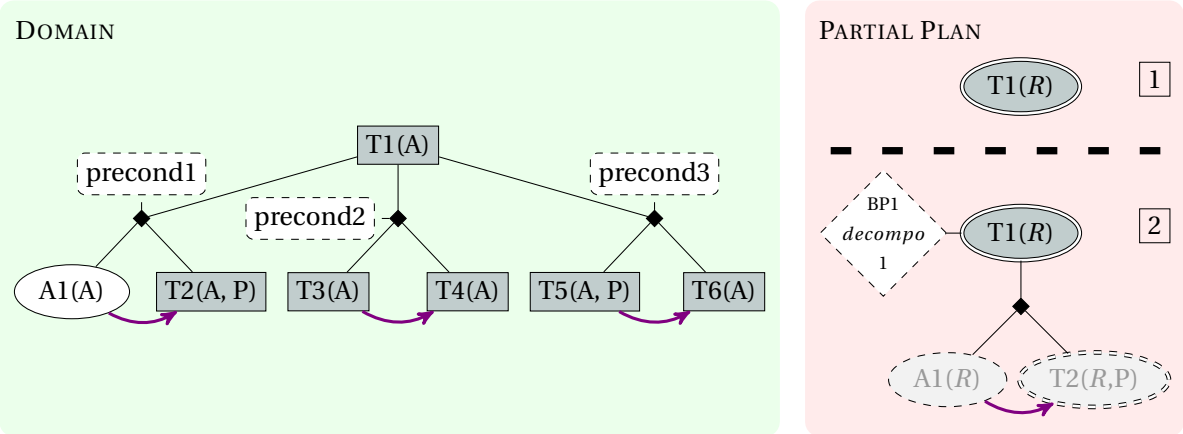


Figure 2.1 – Illustration of the HTN method-selection mechanism. At step 1, let us assume that only conditions "precond1" and "precond2" are satisfied, then one is chosen (randomly) and the other is kept as an alternative of the backtrack point (of type *decompo*). Note the the variable A has been grounded to R, while the variable P (in T2) is still free so a variable binding is necessary.

part (right-hand side) we see two steps -numbered on the right and separated with the dashed line-: (1) the initial state with the task to accomplish, here T1, (2) the decomposition of this task when a method has been chosen. In this particular example we assumed that among all the conditions only "precond1" and "precond2" hold in the initial state, hence only the corresponding methods can be applied. HTN chooses (randomly) a method and uses it to decompose T1. Since there is only one other possible method, it is kept in case a backtrack is needed. Hence a backtrack point is created, with the type *decompo*, and a single alternative.

About the norm we decided the domain tasks are represented with a grey rectangle such as TaskName(Parameters). Each method that matches this task (e.g. task(m) =TaskName) is represented as a black diamond, its preconditions in a small dashed rectangle, and its decomposition is represented under it (the name of the method is not represented in the figure). To represent the precedence constraints, constr(m), we use causal links displayed as the violet arrows (e.g. in the figure A1(A) must occur before T2(A, P)). Usually the "DOMAIN" panel (left-hand side) only contains an extract of the complete domain: the part that is relevant at the planning step depicted.

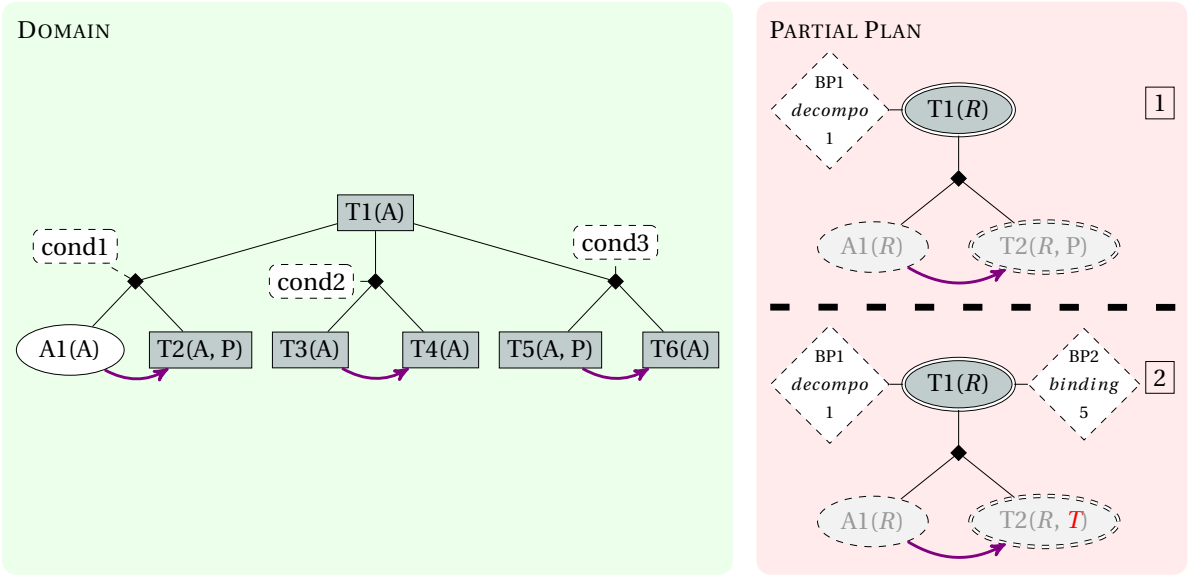


Figure 2.2 – The figure shows how HTN handles a variable binding: T2 was missing a parameter 'P'. In the example we decided there are 6 possible value hence one is chosen[4] and five backtrack alternatives are created in BP2. Note that both tasks A1(R) and T2(R, T) are still greyed-out to represent the fact they still have not been tested: the binding occurs before trying out the tasks.

As the partial plan is built, the right panel will represent the states of the decomposition tree (or partial plan). The actions will be coloured with a colour for each agent, although the preconditions for action A1(R) have not been tested (it has just been added) so it is marked "ready" (grey text and dashed border, as for the task T2(R, T)). A grounded variable (parameter) will be represented in italic

[4]The choice of using 'T' as the parameter value is highlighted in red as are all the changes that are not obvious to see.

(e.g. R) while free variables are kept with the standard font (e.g. P). The choice of using grey ellipses for the tasks (like for T1(R)) in the partial plan comes from the need of being able to differentiate the domain from the partial plan in a glance. Finally when HTN has to make a choice a backtrack point is created with a set of alternatives, to represent them. Therefore we use a box attached to the corresponding task with the following information: (1) BPi the identifier of the backtrack point (useful to understand which backtrack point will be used first when backtracking), (2) the type of the backtrack points and (3) the number of backtrack alternatives that are available.

2.2.2.2 Variable binding

When HTN has to bind a variable it may occur that there are several possible values matching the parameter type, so again as HTN must make a decision some backtrack alternatives are created, see Figure 2.2. The parameter type needed for the parameter is represented in its location: T2(A, P) needs a value of type 'A' and another of type 'P', in our example we chose respectively 'R' and 'T'. While R is given by the task to achieve, T could take another value (we assumed five other possible values). Later in the thesis we may use, instead of the type, a meaningful name as ungrounded parameter if the type is obvious, e.g. Goto(From, To) obviously needs locations for parameters.

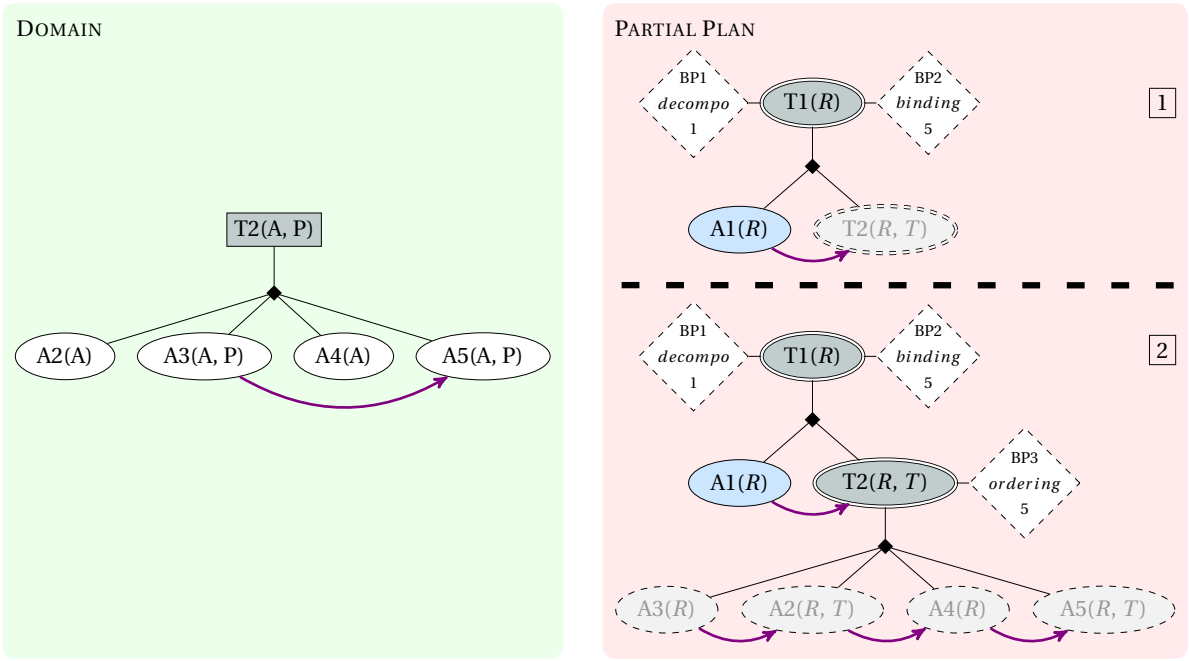


Figure 2.3 – In this step of the example the task T2(R, T) must be decomposed (A1(R) has been tested and is valid). In the domain only one method is available so it is directly chosen (and no backtrack point is needed). However the method does not give a definite order for the action A2, A3, A4 and A5, only that A3 must come before A5 so there is a total of six possible orders. One the orders is used and the planning process can proceed: test that the actions in this order are applicable in the current state (after A1(R) is applied).

2.2.2.3 From partial order to total order

We mentioned before we are using a total-order HTN and need to linearise all the possible task orders when adding some tasks to the partial plan. The Figure 2.3 presents this linearisation process. It is important to note that before decomposing T2(R, T) a test is conducted on the preconditions for A1(R). Indeed a causal link implies that to apply T2(R, T) the action must be possible to achieve. Furthermore when an order is found all the new tasks must be tested.

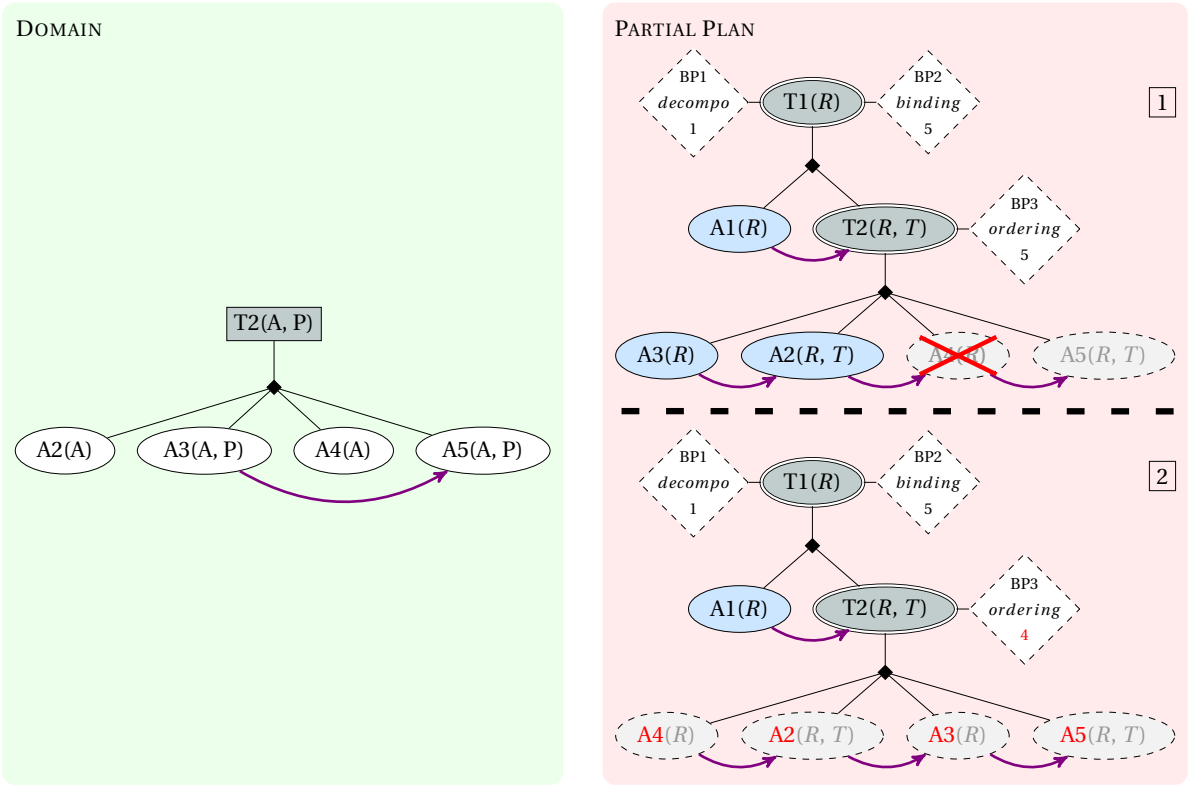


Figure 2.4 – The preconditions of action A4(R) are not valid, so a backtrack is triggered. The closest/latest backtrack point is BP3 (with only four more alternatives left) so a new task order is tried and the planning process continues.

2.2.2.4 Backtrack mechanism

The last HTN mechanism presented is the backtrack, the reason we may need a backtrack are: (1) an action in the partial plan can not be applied in the current state (i.e. its preconditions do not hold), (2) an abstract task has no applicable method (i.e. all corresponding methods have invalid preconditions), (3) a plan has been found and the searches continues for a new one.

In Figure 2.4 the preconditions of action A4(R) are not valid so a backtrack is triggered. As HTN uses a depth-first search, it will go to the last backtrack point inserted and use one of its alternatives

to proceed. If all the orders are not to be valid, hence exhausting BP3, the planner would backtrack even more changing the value used for the parameter (backtrack to the variable binding).

2.3 Conclusion

In this section we motivated the need for hierarchical planning and presented the formalism for HTN planning. Then we illustrated its algorithm by showcasing some of its internal processes. Some choices have been made to present this algorithm, there are indeed several ways to implement it. Additionally we have categorized and illustrated the different types of backtrack. We will introduce a new type in Section 4.4.2 and propose a new plan-search procedure in Section 5.1 that integrates a reasoning about those different backtrack-points types.

We presented this particular instance of HTN because our planner, Hierarchical Agent-based Task Planner (HATP), uses this variant. The order the choices are made while planning does not change the completeness of the planner[5], indeed it solely changes the order the solutions are considered and found. For instance it is possible to switch between considering the decomposition first then the variable binding or the other way around.

The next section presents our planner HATP as it was before the integration with a geometric task planner: it focuses on the actual differences with the generic total-order HTN.

action	$a = \langle \text{name}(a), \text{task}(a), \text{precond}(a), \text{effects}(a) \rangle$
method	$m = \langle \text{name}(m), \text{task}(m), \text{precond}(m), \text{subtasks}(m), \text{constr}(m) \rangle$
backtrack point	$BP = \langle \text{type}(BP) = decompo \mid bind \mid order, \text{task}(BP) = t \rangle$

Table 2.1 – Reminder of the main formal definitions for HTN.

[5]When we talk about HTN completeness from here, it means to find a solution that can be found in the domain. Not in the sense of completeness as in the classical planning, a plan may not be found by HTN if it is not contained in the domain.

CHAPTER 3

THE HIERARCHICAL AGENT-BASED TASK PLANNER, A ROBOTIC HTN PLANNER

The goal of this chapter is to present our HTN planner as it was before the integration with the geometric task planner. Indeed the The Hierarchical Agent-based Task Planner (HATP[1]) differs from a classical HTN planner: it has been designed with the intent to serve as a robotic planner for Human-Robot Interactions (HRIs). To work with humans requires the robots to be socially acceptable: to display certain features taking into account the humans' comfort. For instance correctly distributing the work-load in the group. The first section introduces our formalism of the existing algorithm since it is different from a total-order HTN. The rest of the chapter presents the planner as it was before the work of this thesis. The second section introduces our domain language that was specifically crafted to ease the development of new domains. The third focuses on the built-in features to accommodate the human. Finally the fourth section showcases the integration of our planner in a working robotic architecture.

3.1 Differences with the HTN formalism

As mentioned before, the total-order HTN formalism we introduced in Section 2.2 is already not the classical formalism. It is meant to reflect the choices made in HATP, however as demonstrated it keeps the same properties. The first thing we changed was to switch from classical literals to state variables, exhibited in the first part. Then we show our domain language that is aimed at computer scientists and roboticists. We finish this section by presenting an algorithmic improvement to speed up the planning process.

[1] Its previous name was Human-Aware Task Planner, so it may appear under this name in publications prior to this thesis.

3.1.1 Representing a symbolic state with state variables

Unlike classical planners we do not represent the current planning state s with a set of literals (propositional representation), we are using a state-variable representation. All the elements in the environment are called *entities* and they encode the information for each element. The agents are first-class entities since they act in the plan, this is justified in Section 3.3.1. All the entities have different properties represented as *attributes* of the entities. Those attributes have several characteristics: they can be dynamic or static, they can store a single value (atom) or a set of values and they can be of many types (listed later).

This formalism was first analysed in the SAS+ representation by Bäckström and Nebel (1993). It was demonstrated that a significant number of planning approaches can directly benefit from this representation as shown by Dovier et al. (2007), Helmert (2009) and Dvorak et al. (2013). The main benefit lies in a more compact encoding of a world state using state variables instead of the set of literals. Besides both representation are equivalent, if the state can hold several literals with the same object (e.g. (isOn Table Book) and (isOn Table Bottle)) we can indeed represent it in our formalism with an entity with a set attribute.

In Table 3.1 you can see some examples of our state-variable representations against the propositional representations (here a PDDL instance).

PDDL	HATP
(isAt Robot Loc1)	Robot.isAt = Loc1;
(isOn Table Book) (isOn Table Bottle)	Table.isOn = {Book, Bootle};

Table 3.1 – Comparison on state representation

We argue that the state variables hold another advantage: they are easier to understand for a computer scientist, it resembles the object-oriented language (or even the C-like structure handling). We are demonstrating this likeness further in the following part.

3.1.2 Domain representation

In order to make the domains easier to design we modified a bit the HTN formalism without loosing its properties. The idea is to change a bit the representation of tasks in order to offer an even more intuitive task-network description. Instead of having tasks to which methods and actions are linked with their name() function, we associate directly several decompositions to an abstract task and provide only one action for a primitive task. We are demonstrating, in the next paragraphs, the equivalence between the two representations and that no information is lost.

The first change is that for a primitive action we limit to only one operator. This change is meant to simplify the encoding, in most cases for a primitive task the system has only one way to carry it out. For instance the pick primitive task is usually directly executed thanks to a pick action. For other

cases, where we want several operators for a single primitive task it is still possible to encode in HATP with a simple trick: create an abstract task (with the wished primitive name) that has a decomposition for each possible operator. So the two ways to incorporate operators in the domains are essentially equal.

We recall that a classical HTN method is expressed as follow:

$$m = \langle \text{name}(m), \text{task}(m), \text{precond}(m), \text{subtasks}(m), \text{constr}(m) \rangle$$

In HATP we define a new formalism:

$$m_{hatp} = \langle \text{name}(m_{hatp}), \text{emtpy}(m_{hatp}), \{d_0 = \langle \text{precond}(d_0), \text{subtasks}(d_0), \text{constr}(d_0) \rangle, \ldots\} \rangle$$

where we merge the "name" and "task" functions, and a single method embeds several decompositions d_i under a single name. The empty() function is defined later, let us first focus on proving the equivalence of task decompositions.

Let us define $D_{HTN}(t_u)$ as the set of all decompositions for a task t_u, with the classical HTN:

$$\forall m_i | \text{task}(m_i) = t_u, d(m_i) = \langle \text{precond}(m_i), \text{subtasks}(m_i), \text{constr}(m_i) \rangle, D_{HTN}(t_u) = D_{HTN}(t_u) \cup d(m_i)$$

where $d(m_i)$ represents a decomposition for the task t_u by the method m_i then $D_{HTN}(t_u)$ holds all those decompositions.

We can define the set of decompositions for HATP as well:

$$m_{HATP} | \text{name}(m_{HATP}) = t_u, D_{HATP}(t_u) = \{d_0 = \langle \text{precond}(d_0), \text{subtasks}(d_0), \text{constr}(d_0) \rangle, \ldots\}$$

Then by definition:

$$D_{HTN}(t_u) \equiv D_{HATP}(t_u)$$

Listing 3.1 shows the principle of having several decomposing under a single "method" name. It should be easy to see a resemblance between this form and the object-oriented like languages.

Both in our method formalism and in the listing there is a function called "empty", it aims at representing the possibility of a method being already achieved when the planning process should decompose it. It is expressed as a precondition and if it holds in the current state then the method can be considered already achieved. It is added to the plan but not decomposed and the planning process continues. In the case where a method has a blank empty() clause then it has no effect on the process. Hence this clause only brings convenience since it allows to detect a method that does not need refinement. Furthermore we could express it as well in classical HTN: a method with an empty set of subtasks and that is mutually exclusive (thanks to its preconditions) to all the others for a given task.

Another use for this clause comes with the recursive methods. Indeed it serves as the trivial case to stop the recursion, and again it can be encoded in classical HTN with the same idea as above. Conclusively it can be seen as the goal to reach for the method (abstract task).

```
1  method TaskName(Type1 param1, Type2 param2) {
2      empty{global_condition};
3
4      //First decomposition
5      {
6          preconditions {
7              cond1;
8              cond2;
9              ...
10         };
11         subtasks {
12             ...
13         };
14     }
15
16     //Second decomposition
17     {
18         ...
19     }
20     ...
21 }
```

Listing 3.1 – Abstract version of a task encoded in HATP. In this example two decompositions are visible. In the first decomposition, we can identify the preconditions section and the list of subtasks, the ordering of the tasks is explained later in Section 3.2.2.

To summarise the change in formalism introduced by HATP Table 3.2 lists the mapping between the two. In the rest of the document we will use HATP formalism, and please not that the glossary mentions both versions.

Classical HTN	HATP
an abstract task	a method (sometimes abstract task)
an abstract-task decomposition	a (method) decomposition
a primitive task	a single operator

Table 3.2 – Mapping from classical HTN formalism to HATP formalism.

3.1.3 Cost-based search

Another change to the formalism is the addition of a cost-driven search. To provide this feature we start by associating a cost function (with the prototype: cost : $a \rightarrow \mathbb{R}^+$) to each action:

$$a = \langle \text{name}(a), \text{task}(a), \text{precond}(a), \text{effects}(a), \text{cost}(a) \rangle$$

This function is written as a C++ function, in the domain, allowing complex and advanced computation. In addition, during the planning process, we compute the total cost of the current partial plan as the sum of all the actions it contains. Hence when all the plans are computed we can give the best plan according to the user-provided cost functions.

To further benefit from this cost attributed to plans, when we compute all plans to find the best, we store the current best plan. Then when decomposing a new partial plan, if at any point, its cost

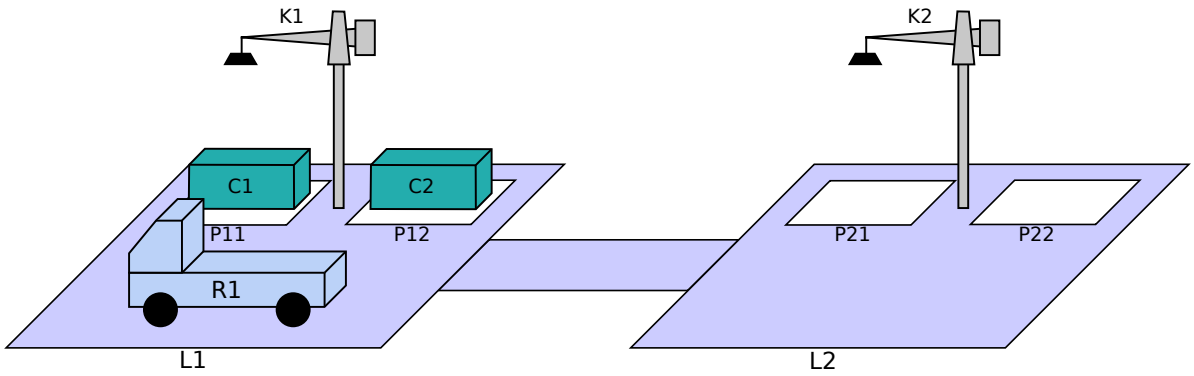


Figure 3.1 – The Dock-Worker Robot problem: the container C1 must be brought to the pile P21 and the container C2 to the pile P22. The cranes K1 and K2 can pick or put container from/in a pile, they can also load or unload the robot. Finally the robot can move from location L1 to L2 back and forth.

exceeds the current best cost then the current partial plan is abandoned, a backtrack is triggered a new partial plan is decomposed. Since the cost functions always return a cost greater or equal to zero, it is not possible that a plan that exceeds the current best plan cost could later becomes better. This mechanism allows a speed-up (see Section 3.2.3 for supplementary use-cases), but it may imply to lose some of the possible plans if not handled with care since some plans are pruned out.

In addition to those features we have a user-friendly domain language which is the topic of the next section.

3.2 User-friendly domain representation

This section gives a glance at the HATP domain language and its special features. For a more detailed documentation please refer to the online homepage[2].

In this section and the next one, we are using the Dock-Worker Robot problem (DWR) as the guiding theme (taken from (Ghallab et al., 2004, Sec. 11.4, p.240)). Figure 3.1 depicts this example. The objective is to transport the two containers C1 and C2 respectively to piles P21 and P22. In order to carry out this task crane K1 must load a container on robot R1 that brings it over to crane K2 which unloads it and put it in the pile, then repeat the process for the other container. In order to prevent this part from being too big only extracts of the HATP domain are presented, however the complete domain can be found in Appendix B.

3.2.1 Representing and comparing entities and states

As mentioned above, we describe the environment with "*entities*", one for each element. Besides we are using a state variable representation. But to ensure that the states and the entities manipulation are consistent we are using entity-type enforcement during any operation.

[2] https://www.openrobots.org/wiki/HATP

3.2.1.1 Defining new entity types

It is possible to define new types and because the agents are first-class entities the Agent type is already defined. You can see such definitions in Listing 3.2 from line 2 to line 4.

```
1  //Agent is already declared
2  define entityType Location;
3  define entityType Pile;
4  define entityType Container;
5
6  define entityAttributes Agent {
7      static atom string type;
8      dynamic atom Location at;
9      dynamic atom Container carry;
10 }
11
12 define entityAttributes Location {
13     static set Location adjacent;
14     dynamic atom bool occupied;
15 }
16
17 define entityAttributes Pile {
18     static atom Location attached;
19     dynamic set Container contains;
20     dynamic atom Container top;
21 }
22
23 define entityAttributes Container {
24     dynamic atom Location in;
25     dynamic atom Container on;
26 }
```

Listing 3.2 – Extract from the DWR domain: defines the new types Location, Pile and Container, then describes their attributes.

After defining the new types, we must define their attributes. As aforementioned, an attribute has properties: it can be static or dynamic, which simply tells the planner whether that value can be changed or not while planning. An attribute then can be an atom to store a single value or a set to store several (e.g. an atom can be the position of an agent, while a set can list what a pile can store). Finally a type is given, it can be a common type among: number (floating point), string or boolean, or it can be any entity type defined earlier.

For instance from line 6 to line 10, we describe the agents as having a type (stored as a string because enumerations are not implemented). Then we also tell that an agent is at a certain Location. Finally the agents can carry a container (lifting it or transporting it) so a last attribute is created for that. The meaning of the other attributes should be understandable from their name.

3.2.1.2 Modifying the entities

Once that all the entities and attributes are defined we can create, modify and test the symbolic state easily. The language offers the classic assignation and comparison constructs for the atomic attributes, and their form resembles the usual programming languages. For the set we propose an operator to add "<<=", to remove "=>>" and to test the presence ">>" or absence "!>>".

```
1    //Creations
2  R = new Agent;
3  K1 = new Agent;
4  L1 = new Location;
5  L2 = new Location;
6  P11 = new Pile;
7  C1 = new Container;
8  C2 = new Container;
9
10 //Initialisations
11 R.type = "ROBOT";
12 R.at = L1;
13
14 K1.type = "CRANE";
15 K1.at = L1;
16
17 L1.adjacent <<= L2;
18 L1.occupied = true;
19
20 P11.attached = L1;
21 P11.contains <<= C1;
22 P11.top = C1;
23
24 C1.in = L1;
25 C2.in = L1;
```

Listing 3.3 – Extract from the DWR domain: create the initial symbolic state.

Listing 3.3 exhibits how the initial state is created. Line 2 shows how to add an entitiy, here an agent, to the environment. When all entities are created we can use them to build the initial state by filling in their attributes. It is important to note that we do not fill in every attribute because we are using the close world assumption, i.e. when a value is not specified it is considered empty (and not unknown). For instance the robot carries nothing since we do not have a line about R.carry. Line 11 tells that the robot is indeed a robot, this is used later in the tasks preconditions to ensure that the agent given to a task (such as move) is coherent with the action to carry out. Finally Line 21 illustrates how an object is added to a set.

Besides those classical constructs, the language allows two advanced constructs. The first, IF, enables to have conditional effects: if the condition holds in the current state only then its effects are taken into account. The second is FORALL, it changes all the entities of a given type that also correspond to its predicates.

3.2.1.3 Predicates on a symbolic state

The previous example shows how to create the initial state, using the same construct it is possible to write the effects of the actions. We describe here how the predicates used in preconditions are formed. This part starts by giving a simple example then advances to some interesting constructs but not all are presented. During this thesis some work has been done to fix implementation issues of the preconditions and is presented in Appendix A.2.1, which also list all possible constructs for predicates.

Listing 3.4 displays the preconditions for the Take action, where a crane K picks a container C from a pile P all occurring at location L. We can see that most of the preconditions are easy to understand,

it starts by checking that K is indeed a crane at the right location and that it is empty. Then it controls that the container is at the top of the pile (that also must be at the right location). Line 5 is particular and demonstrates the construct to test that an element is in a set, here that the container is in the pile.

```
1 K.type == "CRANE";
2 K.at == L;
3 K.carry == NULL;
4 P.attached == L;
5 C >> P.contains;
6 P.top == C;
```

Listing 3.4 – Extract from the DWR domain: preconditions of the take action.

In addition to those straightforward constructs, our language proposes some more complex namely EXIST, FORALL and evaluable predicates. As their name suggests it, they respectively allow to test that there exists an entity with certain properties or that all the entities of a certain type matches some conditions. The latter allows to call an external (C++) function to do advanced computation on the symbolic state and to return a value that can be used in a predicate. Again more details on those constructs are given in Appendix A.2.1.

3.2.2 Hierarchical tree representation

In the previous Listing 3.1 we prove that an HATP method corresponds to a classical-HTN abstract task and can have decompositions. The following Listing 3.5 demonstrates how to actually write such method. The Setup method is responsible for picking up a container from a pile and putting it on the robot. First of all we can see Line 2 that the goal is indeed to have the robot carrying the container. The preconditions start by checking that there exists a pile (at the current location) that contains the container, then that there is a crane at the current location that does not hold anything, to finish by ensuring that the agent responsible for the transport is a robot (and that it is at the desired location as well).

```
1  method Setup(Container C, Agent R, Location L) {
2      empty {R.carry == C;};
3      {
4          preconditions {
5              EXIST(Pile P, {P.attached == L;}, {P.top == C;});
6              EXIST(Agent K, {K.type == "CRANE";}, {K.at == L; K.carry == NULL;});
7              R.type == "ROBOT";
8              R.at == L;
9          };
10         subtasks {
11             P = SELECT(Pile, {P.attached == L; P.top == C;});
12             K = SELECT(Agent, {K.type == "CRANE"; K.at == L; K.carry == NULL;});
13             1: Take(K, C, P, L);
14             2: Load(K, R, C, L)>1;
15         };
16     }
17 }
```

Listing 3.5 – Extract from the DWR domain: the Setup task picks the container and loads it on the robot.

The subtask block describes the decomposition and the variable bindings. Since the Setup method knows only the container, the robot and the location from its parameters we have to find a crane and a pile. So it starts by binding two variables with the same constraints as the one that appeared in the preconditions. Once the variables are bound we can give the decomposition: begin with a Take task followed by a Load (at this point it is not possible to know whether they are methods or actions). The ordering is given using the ">1" symbol which reads as Load must occur after Take. Although we only allow precedence constraints, they are not mandatory so it is possible to express complex partial orders.

Since the description of actions is really similar to the formalism we do not describe them here, but examples can be found in the complete domain in Appendix B.

3.2.3 Control of the planning process

The variable bindings used in Listing 3.5, at Line 11 for instance, only specify the constraints for an entity to be valid. Nonetheless our language offer some control over the binding: we can specify a function to sort the entities or ask to bind the value only once and prevent any backtrack point on this choice.

The goal of the first binding, called SELECTORDERED, is to benefit from the plan pruning based on the cost. Since it sorts the entities we may expect the first plan to be better and that the other plan will be pruned out, reducing the planning time. For instance consider that a robot has to go to a location to perform some tasks, if all the available robots are sorted using their euclidean distance to the goal the first robot tried should be closer to the goal. When all plans are computed to find the best, if another robot is tested it may be sufficiently far away to already exceed the best plan cost an be rejected, saving the time since all the following tasks are not tested.

The other binding, named SELECTONCE, allows to stop after the first value is found. The best use for it is when a recursive methods works on a set where the order of the elements does not matter: for instance emptying a tray on a table. Using it may greatly reduce the number of computed plans to find the best one. Indeed to recursively handle a list while keeping the backtrack alternatives corresponds to a factorial number of plans to try (e.g. first choice is done over 10 objects, then 9, then 8 and so on, equivalent to 10! which is a very big number).

3.3 Planning for the humans and robots

This section is centred around the consideration of the human in the planning process. It starts by presenting the explicit representation of the agents to produce plans more suited for execution of cooperative plans. Then it presents some ways to filter the solutions to keep the ones that exhibit socially-acceptable behaviours. And finally a bit of belief management is introduced.

3.3.1 Represent the agents

Since actions must be carried out by the agents of the system, we issue a requirement for domain developpers: each action must have at least one agent in its parameter. Then during the planning process, HATP assigns the actions to the corresponding agents which allows it to build streams of actions. Thus a stream is created for each agent in the domain, causal links and joint actions are used to synchronise those streams. A joint action corresponds to several agents acting together in a tightly cooperative fashion (e.g. transporting an object together, exchanging an object), since those actions require several agents it obviously needs them all to be ready at the same time hence enforcing a synchronisation.

On the other hand the causal links are inferred from two sources: the hierarchy in the partial plan and the "resources" usage. The former gives most of the causal links, but some of the links are not kept in order to increase execution efficiency. The latter is in fact computed from the entities: if an action modifies an entity attribute and then another action in its preconditions performs a test on the same entity attribute then we consider that the two actions share a resource and a causal link is added. We are using the DWR example, Figure 3.3, to show such streams

3.3.2 Toward being socially optimised

In order to obtain solution plans that are more suitable for interacting with humans, ways to filter out some solutions have been developed. When a plan has just finished to be produced a set of *social rules* are applied to penalize it if it does not display social behaviours. There are many social rules implemented and they can be activated or not depending on the context, we present here only the most interesting ones.

Prevent occurrence of forbidden states and forbidden sequences The goal of this rule is to allow the domain expert to specify some states or sequences that should not be reached. This is useful mostly to ensure that the humans do not end up in situations that would affect their comfort.

Balance the effort This rule controls the way the effort is shared by the agents, allowing to adapt the effort in the group. For instance it allows to balance it evenly or to put as much load as possible on the robots to reduce the quantity the humans must do. A fair distribution of the work load could be used in an industrial context. On the other hand, asking the robots to do most of the work, would be more suited for elderly care. The main notion here is effort, not number of actions, indeed it is possible to have many simple actions to do as opposed to one single complex action.

Avoid wasted time This rule can work conjointly with the previous because it penalises plans where an agent spends too much time idle. This usually represents plan where an agent has to wait for many actions from another agent. This rule prefers plan where the time spent without acting is reduced.

Control the intricacy This last rule allows a finer control than the previous one. Indeed it is focused on the number of times an agent actions depend on the completion of the other agents actions. It simply counts the number of causal links that go from a stream to another and the number of joint tasks. The principle is that the more intricacy the more impact an action failure would have on the overall plan. If an agent fails to perform an action that other agents depends on then the execution layer must ask for a completely new plan. While plans with less intricacy have more chance of displaying almost parallel set of actions which offers a lot more flexibility: the actions may be reordered by the execution layer, a local action repair may be possible (change only few actions in the stream without threatening the rest of the plan).

To conclude this section it is important to understand that those rules have no effect on the completeness of the planner. It only changes the definition of the best plan to be more acceptable by humans co-workers, which can be quite different from the "naive" best plan HATP may produce without such rules. It has been demonstrated in the work from a predecessor: Alili et al. (2008, 2009).

3.3.3 Put the robot in the human's position

Whereas it is easy to share the information in a centralised robotic architecture, it becomes a problem when working with humans. In order to know which information must be shared, we need a model of their knowledge of the current world state. That way it is possible to know if they are missing information when the planner expects them to perform certain actions. For instance if the robot knows where is the next object the human must pick but the human can not see it then the robot must inform it of the location.

To tackle this issue we are using a belief model to represent what the robot thinks the human knows (it can be wrong but the execution layer should handle it). It is then used to detect: (1) when there is a divergent belief to correct the incorrect piece of information, (2) to detect when the human may not know the information in order share it and (3) when the robot needs to know something and it must ask the human to lift the uncertainty. This work is close to both the planning-and-acting field and the probabilistic robotics (working with belief states). Even though it is just a step towards those fields to ease the development of the execution layer, it allowed to obtain some interesting results. Many papers were published to introduce this human-aware aspect, some of the papers are Alami et al. (2011), Clodic et al. (2007), Guitton et al. (2012a), Warnier et al. (2012).

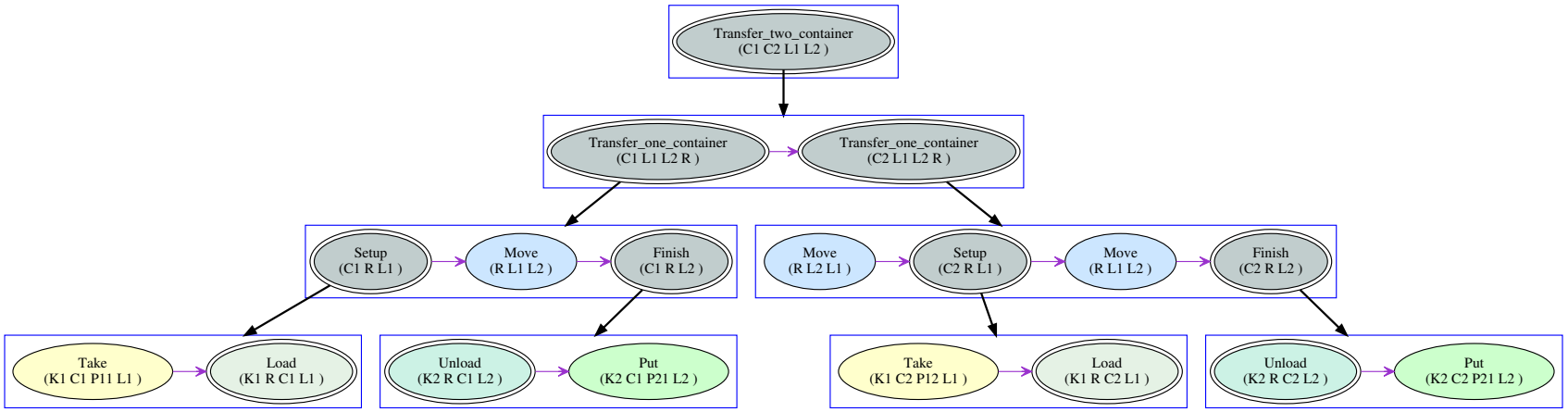


Figure 3.2 – The decomposition-tree solution for the DWR problem. The blue rectangles represent the decomposition of the methods. We can see that all the actions are not linked with causal links as opposed to the following figure where the actions are organised into streams.

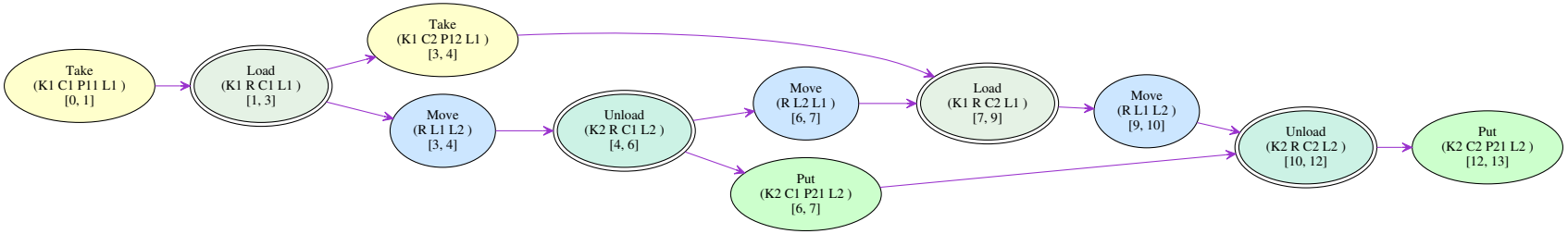


Figure 3.3 – Streams of actions for the DWR problem. Yellow actions belong to the crane K1, the blue actions to the robot R, and the plain green to the crane K2. The Load and Unload actions (with some other shades of green) represent the joint actions. It is important to compare the two representation, for instance the second Take action from K1 is ordered right after it has loaded the robot, while in the tree view we can trace out a series of causal links that would put it instead after the second Move from R. It illustrates the fact that some causal links are removed to allow a more efficient action allocation.

3.4 A planner that works

The last section is just a glimpse at the details of the implementation of HATP and its integration in a complete robotic architecture. A more thorough presentation of some parts of the implementation can be found in Appendix A. This small section demonstrates how HATP is a "planning laboratory".

3.4.1 A C++ implementation

Because HATP concepts are based on object-oriented programming, its syntax and semantics lend themselves to being implemented almost directly using "raw" C++ data structures such as classes and complex data types. More specifically, unlike classical HTN planners which model a state as a set of first-order logic, an HATP state is implemented as a data structure representing the entities as C++ objects.

The previous sections show that the HATP domain language itself relies on object-oriented-like structures, then to internally represent them as C++ structures makes a seamless transition from the domain to the internal structures. This allows a better understanding of the internals of HATP and ease the development of new extensions and the integration with other systems.

Although the implementation has not been extensively optimised, it still allows to produce plans in a time that permits the robot to answer in a reasonable time. In de Silva et al. (2015), we prove that its planning time is comparable to JSHOP (the Java implementation of SHOP).

3.4.2 A module in a robotic architecture

As mentioned before, HATP has been used for many years in the LAAS architecture to work in HRI scenarios. The following papers presented this architecture: Alami et al. (2006a), Ali et al. (2009). See the table at the end of this chapter for the complete list of past and recent publications featuring HATP.

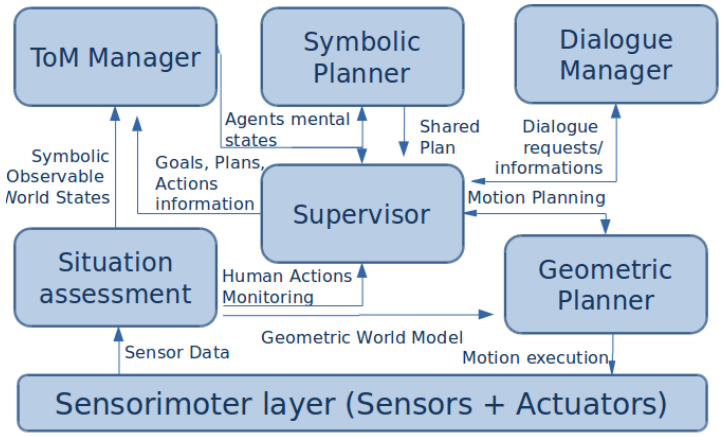


Figure 3.4 – Architecture with Theory of Mind and HATP: the system must reason about the user knowledge and adapt the execution of the shared plan to the estimated mental state of the human.

In more recent years, it has been integrated in even more scenarios, mostly for European projects. The most notables are Saphari (cooperation of humans and robots in industrial scenarios), Arcas (multi-UAV building assembly, presented later) and Spencer (a robot that guides people in an airport). It implies that it has been integrated with a variety of other modules: databases, different execution supervisors, different middlewares and so on.

For instance in Devin and Alami (2016), HATP has been used to elaborate plans that are meant to be executed in collaboration with a human. Since there may be some unexpected situations that would cause the human to miss some pieces of information on the current situation a module to manage this Theory of Mind (ToM) has been developed. Figure 3.4 presents the architecture, where HATP is the symbolic planner. In this system, HATP retrieves the initial state from the Situation assessment module through the Supervisor but also the estimated agents mental state from the ToM Manager. It sends the solution plan to the supervisor in both the solution tree form and the streams of actions. The Supervisor then complete this plan with the trajectories computed by the Geometric Planner and can finally execute it on the robot while monitoring the human's actions to ensure there is no deviation.

In addition to the different integrations, some tools were specifically designed to help work with HATP. The first is a software to display the best plan and some statistics on the search, in a way convenient for both domain experts and the other persons interacting with HATP. Figure 3.5 presents a screenshot of this interface.

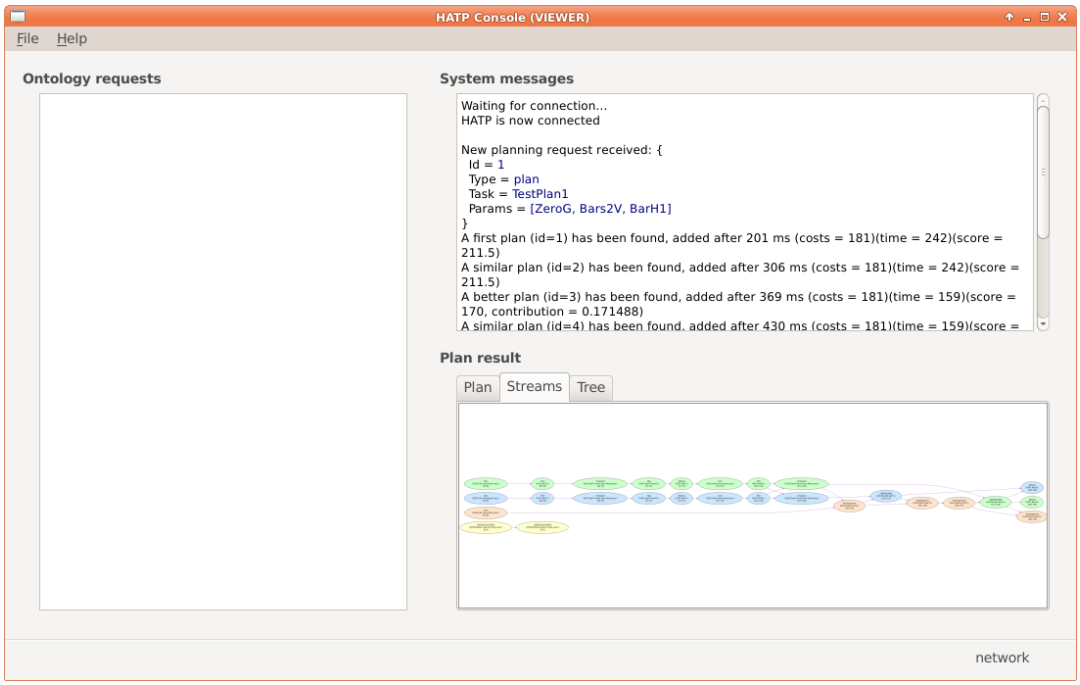


Figure 3.5 – Screenshot of the interface to display the plans found by HATP. Note that the bottom-right panel has several tabs that display a plan in different forms, respectively as a formatted text, the streams (current view) and the decomposition tree.

3.5 Conclusion

In this chapter, we have presented the Hierarchical Agent-based Task Planner as it was before this thesis. It is the stepping stone for the rest of this document.

HATP is an HTN planner improved to be more suitable for robotics application, more specifically in the context of Human-Robot Interactions.

HATP has been improved both internally, enhancing the HTN search, and externally by providing a user-friendly language. First the search is faster thanks to a cost-driven search: plans that exceed the current best plan cost are pruned out, offering a speed-up in planning. In addition to being faster the planning process has also been improved in order to explicitly represent the agents.

As the plan is built, the actions it is composed of are split into streams, one for each agent. This makes it more convenient for the execution layer but also offers an access to filter out plans that do not exhibit proper social etiquette. HATP proposes to apply social rules to favour the plans that are the most suited for HRI.

Last but not least, a lot of effort has been given to the design of a language for easy domain development. It has been inspired from the object-oriented languages hence making it easy to understand and manipulate for computer scientists. Even though it is easier to manipulate neither does it lose completeness nor it prevents control over the planning process. It is indeed possible to change the behaviour of the planner, by changing the hierarchy or by using the different means available to control the variable binding.

HATP must be seen as a planning laboratory, extensible while staying efficient enough for real-world applications as we demonstrated through the numerous works done by the predecessors and HATP integration in the LAAS architecture.

To reflect the strong orientation of HATP toward multi-agent problems, and because this thesis aimed at developing new domains such as multi-UAV: HATP has been renamed Hierarchical Agent-based Task Planner[3]. It keeps its acronym but emphasis the broader range of problem it can tackle.

action	$a = \langle \text{name}(a), \text{task}(a), \text{precond}(a), \text{effects}(a), \text{cost}(a) \rangle$
method	$m = \langle \text{name}(m), \text{emtpy}(m), \{d_0 = \langle \text{precond}(d_0), \text{subtasks}(d_0), \text{constr}(d_0) \rangle, \ldots\} \rangle$

Table 3.3 – Reminder of the old formal definition for HATP.

[3]This thesis was funded with the Arcas European project which focuses on Unmanned Aerial Vehicles and a copy a HATP was created in case some major changes were to happen. This copy is called Multi Aerial-robots Task Planner (MATP) but in the end is the same planner as HATP.

3.5.1 List of publications on HATP

Many of the work published on HATP are centred around its multi-agent capabilities and its social-cost filtering. They present the shared-plan structure it utilises to permit cooperative work between humans and robots: Alami et al. (2006a), Alami et al. (2006b), Clodic (2007), Clodic et al. (2007), Montreuil et al. (2007), Montreuil (2008), Clodic et al. (2009), Alili et al. (2009), Alili et al. (2008), Ali et al. (2009), Alili et al. (2010), Alili (2011), Pandey et al. (2011), and Ali (2012)

Later most of the work was more focused on belief management in the context of HRI: how to represent the human co-worker mental state and understanding of the current state and plan. And how to use this model to decide the best course of actions to lift any ambiguity or lack of knowledge: Alami et al. (2011), Warnier et al. (2012), Warnier (2012), Guitton et al. (2012a), Guitton et al. (2012b), Fiore et al. (2015) and Devin and Alami (2016).

Then our more recent work on integrating symbolic task planning with geometric task planning: de Silva et al. (2013a), de Silva et al. (2013b), de Silva et al. (2014), Lallement et al. (2014) and Gharbi et al. (2015).

Finally some papers are more focused on presenting a particular use case for HATP, whether it is industrial scenario with Alami et al. (2014), or dialogue interactions in Milliez et al. (2015), or in a combination with learning tools in Renaudo et al. (2015). Last but not least, a full paper has been dedicated to formalising and analysing HATP language: de Silva et al. (2015).

CHAPTER 4

COMBINING SYMBOLIC AND GEOMETRIC PLANNING

This section focuses on the combination of our symbolic planner with a geometric planner. This work was held in tight cooperation with Mamoun Gharbi, who brought the knowledge on the geometry, he already defended it in Gharbi (2015). First we justify the need for such integration and define the vocabulary. Then we present the related work. We proceed by a presentation of the preliminary work done by our predecessors. Section 4.4 demonstrates the formalism and algorithms we are using. It is followed by a section presenting a way to improve the search using the knowledge an expert can provide.

4.1 Why combining symbolic and geometric planning

One of the main issues of symbolic task planning for robots is its inability to reason about the geometry of the real world. A motivating example: a robot must put some objects on a table, without geometric reasoning the robot has no way to know if the table will be big enough for all the objects. A naive approach consists in discretizing the geometric world state and work only at symbolic level with the (finite) set of discrete positions. This is tedious and also very limited since it does not allow for any fine manipulation, indeed if the objects have complex shapes it will not be sufficient. A solution could be to try the objects placements in a simulator to verify if they can fit. Since we also need to assess the trajectories feasibility, a simple simulator is not enough. We go further, using a motion planner, that allows to also ensure that the robot can place the object: the necessity for this test lies in the fact that each time an object is moved it changes the topology of the environment and may prevent the *feasibility* of any further action in the same area (while still allowing the placements).

In order to address this problem, combining task and motion planning has been of great interest in a number of studies during the few last decades. Before presenting the related work we must spend a bit of time defining the terminology. To this point we have presented our task planner HATP, the

"symbolic planner". It is part of the symbolic layer (or symbolic level), we present the other part of this layer along this chapter. At the other end of the system we have the motion planner (or path planner), a module of the geometric layer (or geometric level, or simply geometry). Furthermore, when a formalism refers to the symbolic layer (resp. geometric layer) we use a symbol to denote its origin, for instance a symbolic entity is E_s (resp. geometric entity is E_g). This definition is necessary since the geometric layer may produce symbols to provide information to the symbolic layer.

Along the years the problem was given various names, such as "manipulation planning" in Alami et al. (1989), "hybrid planning" in Guitton and Farges (2009b), or "Combining Planning and Motion Planning" (CPMP) from Choi and Amir (2009), or "Task And Motion Planning" (TAMP) in Lozano-Pérez and Kaelbling (2013) and its variants ITMP for Integrated TAMP in Nedunuri et al. (2014) and Hauser and Latombe (2009), STAMP for Simultaneous TAMP in Şucan and Kavraki (2012) or finally CTAMP for Combined TAMP in Lagriffoul et al. (2014). We propose Symbolic-Geometric Planning (SGP) since we do not only plan tasks and motions we also reason on the symbolic and geometric consequences, their intricacy and how they interfere one with another.

4.2 General consideration, basic formalism, and notation

This section introduces the general concepts and just enough formalism to offer a common ground to comment the related work in the next section. The very first and probably most important topic concerns the state representation in each layer -symbolic and geometric-. For most of the systems the symbolic layer represents a symbolic state s in S using literals (or state variables), while the geometric layer works with continuous search space, and a state is called a world state $\mathcal{W}$ (or configuration) in Q. It is important to point out that many different configurations can be interpreted as a single symbolic state, for instance any position of an object $O1$ on top of a given table T (its centre of mass can have any x and y in the bounds of the table surface) would produce a unique literal $O1.isOn{=}T$ (or *(isOn O1 T)* using classical representation).

Figure 4.1 shows the transformation from an action a into the set of trajectories $\{traj_0 \dots traj_m\}$. Another aspect not depicted in this figure is that each trajectory depends on an initial state. Most of the time those intermediate world states are not fully specified at the symbolic level and some decisions must be taken by the geometric layer, such as gripper position for a grasp, object placement for a place action, and so on. We call *alternatives* the different possibilities to refine a symbolic action into different trajectories by changing their parameters (initial/final configuration, and so on).

When the trajectories are found, it is possible to extract literals that represent the resulting world state in the symbolic level: the symbolic planner reasons about properties and not about the configurations. When those literals from the geometry are combined with the one from the symbolic action specification we can build the new symbolic state s' and proceed with the planning process. However for a single symbolic action we may have different geometric alternatives, henceforth different final states and shared literals, which may create different finale symbolic states.

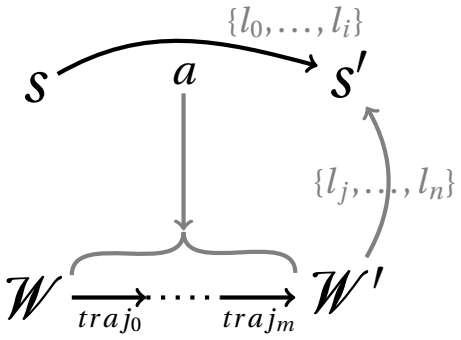


Figure 4.1 – The link between the symbolic and geometric layers, An action a changes the symbolic state s into s' with the literals $\{l_0,\ldots,l_i\}$. However it may be necessary to refine it into a set of trajectories $traj_0 \ldots traj_m$ that turn the geometric world state $\mathcal{W}$ into $\mathcal{W}'$. It is then possible to compute the additional set of literals $\{l_j,\ldots,l_n\}$ to represent the outcome of the geometric counterpart of the action and enrich s'. Since there can be several instantiation of the action a in the geometry we may have different trajectories and choice. This creates different finale world states and may cause the shared literals to be different as well, creating different final symbolic state s'.

s in S	Symbolic state (in the state space)
$\mathcal{W}$ in Q	Geometric world state (or configuration)
a	Symbolic action
$traj_i$	Geometric trajectory (path with time)
O.isOn=T	Literal from symbolic layer, or symbolic literal
O.isOn=T	Literal from geometric layer, or shared literal

Of course there are several approaches to combine symbolic and geometric planning. This section purpose is to introduce some common terms and an intuitive way to translate from a symbolic action into geometric trajectories. We give more details on our way to do this transformation later in this chapter, how symbols are given to the geometry and so on. We show in the next section that there are actually different ways to do it.

4.3 Related work[1]

Different approaches were proposed, Schüller et al. (2013) distinguish four different strategies: "(i) low-level checks are done for all possible cases in advance and then this information is used during plan generation, (ii) low-level checks are done exactly when they are needed during the search for a plan, (iii) first all plans are computed and then infeasible ones are filtered, and (iv) by means of replanning, after finding a plan, low-level checks identify whether the plan is infeasible or not; if it is

[1] This work was realised with Mamoun Gharbi, hence this related work is the result of a cooperative work and was first presented in his thesis Gharbi et al. (2015).

infeasible, a new plan is computed considering the results of previous low-level checks". While this allows to sort the publications we propose another categorisation which reuses the same strategies but add some categories. It is indeed necessary to define categories because they propose different strategies but the main point is that they address different classes of problems that could not be solved by all other categories. For instance some problems only require to compute motions that do not change the position of the objects, whereas some problems involve intricate manipulation, some other problems need subtle state description and so on.

Our categorisation is based on the fact that the problem is in fact the composition of a symbolic state[2] s from S and of a geometric configuration[3] $\mathcal{W}$ in Q. Since a trajectory can modify the symbolic state, we can define the next state as $s' = \gamma(s, \mathcal{W}, a)$. In other words when we decide of an action it modifies both the symbolic and geometric states.

Thanks to this definition we see that we have different possible decompositions for $S \times Q$:

Symbolic layer calls the geometric layer In this situation, the symbolic planner performs the search, and verifies the plan by asking the geometric level about the feasibility. (This category groups the strategies (ii), (iii) and (iv) from Schüller.)

Geometric layer calls the symbolic layer In this case, the geometric planner knows the possible solutions and uses the symbolic layer to determine which ones to explore or choose. (It corresponds the strategy (i).)

Sample in the compound state The search space here is a compound space between the geometric and the symbolic spaces, the search is done at both levels simultaneously. (This category does not exist in Schüller et al. (2013).)

The rest of this section is devoted to presenting the state of the art using the categorisation proposed.

4.3.1 Symbolic layer calls the geometric layer

This approach can be further refined into three sub-categories: (1) all the symbolic plans are computed and the geometry is used to find a feasible one, (2) a first complete symbolic plan is computed then it is tested in the geometry or (3) the symbolic planner calls the geometric layer as it builds the plan. Each of those sub-categories are presented in the following sections.

4.3.1.1 Find a geometrically feasible plan among all symbolic plans

The principle of this approach is that the symbolic planner computes all the possible symbolic plans, and provides them to the geometric layer that tries to find one that is indeed feasible. Şucan and

[2] We are here going back to the classical planning representation for the related word section. Hence S is the set of all the symbolic states, and from a given state s we can compute the next state s' when applying an action a: $s' = \gamma(s, a)$.

[3] A configuration $\mathcal{W}$ completely describes the environment by giving all the positions of each object and the configuration of each joint in the agents. Q is the set of all the possible configurations.

Kavraki (2011) present a solution where, given a list of possible plans they are able to find a feasible set of motions that fulfil the given symbolic goal. Şucan and Kavraki (2012) extend this approach by introducing uncertainties, and use a Markov Decision Process to guide the search. Lagriffoul (2013) propose a different way to solve the problem: they argue that part of the geometric reasoning may be endowed to the symbolic planner. They use a HTN where they broke the geometric actions into basic primitives, to find all the possible plans, then they use a geometric reasoner to test the geometric feasibility of the plan using what is called geometric backtracking: they commit to the symbolic plan and only reconsider the geometric choices.

We show in Section 4.4.2 that a geometric action may have multiple alternatives. Let us consider a case where a robot has to place two objects on a table, if the position chosen for the first object is not correct it may leave not enough space for the second object to be placed. Another position of the first object can allow the two to be placed and the plan to succeed, therefore although the symbolic plan is valid, the geometric instantiation of an action is not valid and a backtrack is required. Geometric backtracking occurs when the geometric reasoner fails to find a solution for an action, the geometric layer then keeps the symbolic plan but tries different alternatives for the actions that were already proven feasible. The idea is that by changing an action that succeeded but prevent another one, the outcome can be sufficiently different to allow the next actions. Usually there is a limit on the maximum number of geometric alternatives to be tried for a specific geometric action, or *branching factor*, in order to prevent an explosion in the number of solutions to compute.

4.3.1.2 From a first symbolic plan find a geometrically feasible solution

The classical approach in robotics consists in computing the symbolic plan, then for each action ask a motion planner to compute the trajectory. If the complete plan can be transformed this way then it is executed, otherwise a new symbolic plan is computed. This category presents the approaches of SGP that is the closest to the classical approach. However they differ from it since they are explicitly taking into account the geometry here. The idea is to prune out impossible symbolic plans right from the start of the planning process.

Lozano-Pérez and Kaelbling (2013) build a plan skeleton using symbolic planning, it contains geometrical constraints hence formulating the problem as a Constraint Satisfaction Problem. Then they use a general solver to test the plan geometrical feasibility. In Srivastava et al. (2013) case, once they find a task plan, they try to plan the geometric actions, and if they fail an error is returned to update the symbolic state and a new task plan is computed. Caldiran et al. (2009a) and Caldiran et al. (2009b) propose a different solution where they use an action description language (namely $\mathscr{C}+$) to provide the robot with a high-level reasoner capable to find a complete symbolic plan, then they extract from it the collision free trajectories. When they encounter a problem (collision) they report it to the reasoner and a new symbolic plan is computed where they try to extract the trajectories again. They demonstrate with an example where two robots move object in a 2D grid, and propose another example in Haspalamutgil et al. (2010): the tower of Hanoi problem. Erdem et al. (2011) keep

nearly the same framework but use, in place of the action description language, a causal reasoner to find the symbolic plans. If the geometric resolution fails, it changes the planning problem by adding constraints to the causal reasoner in order to take the causes of failure into account. As before, they use two robots to move an object and extend it again to solve the tower of Hanoi in Havur et al. (2013).

In this sub-category, some of the approaches also rely on geometric backtracking. Srivastava et al. (2014) present an interface between a task planner and a geometric planner where, once a symbolic plan is computed, they use geometric backtracking to test its feasibility. They have limited number of alternatives for each actions and if they exhaust all of them and still fail to find a collision free trajectory, the geometric reasoner informs the symbolic planner about the infeasible action and the reasons for its failure. The task planner then changes the part of the plan after the last successful (feasible) action. Lagriffoul et al. (2012) also use geometric backtracking on a complete plan, but they introduce the notion of constraints on the interval bounds to speed up the search. Once they get the symbolic plan, they use it to define constraints on the robot configuration at each step, starting from the last step. These constraints reduce the geometric search space of each action making the number of geometric backtracks drop. Lagriffoul et al. (2014) extend this approach by adding constraints over several degrees of freedom and exhibit a study of the time complexity of their algorithm. Dearden and Burbridge (2013) also compute the complete symbolic plan before computing the geometry, then they map the symbolic states with geometric ones, starting from the final state. Finally they try to find trajectories between the states. If a trajectory can not be found, a geometric backtrack is triggered in order to change the symbolic-geometric state mapping. This mapping is learnt through a set of training data in the form of geometric states labelled with the predicates that hold true.

4.3.1.3 Ensure geometric feasibility while computing the symbolic plan

This sub-section contains approaches where the geometric layer is called during the elaboration of the symbolic plan, for instance after a new task is added to the plan in order to ensure its geometric feasibility “on the fly”. The idea is to prevent exploring symbolic plans that are infeasible because of an early action.

Dornhege et al. (2009) introduce the notion of semantic attachments, in the context of SGP, which corresponds to external procedures able to either evaluate if a condition holds or compute the numerical value of a state variable. The condition validation can be used in an action preconditions and computes the feasibility of a motion corresponding to the action. The state variable computation is used to retrieve the new world state from the geometry. Dornhege et al. (2010) present a study about soundness and completeness of their approach in addition to multiple examples and relevant results. Dornhege et al. (2012) introduce the possibility of using heuristics during the search by enabling the semantic attachments to only return an evaluation of their computation and propose the use of different off-the-shelf task planners able to use these heuristics, e.g. Fast Forward from Hoffmann and Nebel (2001) or Temporal Fast Downward from Eyerich et al. (2009b). Hertle et al. (2012) propose a new planning language: Object-oriented Planning Language, where the tasks description are written

in an easy object-oriented-like form (similar to C++) and which can handle semantic attachments. In addition to describing the domain, their language is also used to produce a domain-specific interface that allows an easier (and type-safe) integration of external modules. The latest extension added by Dornhege et al. (2013) to this work consists in caching the external procedures return values and states in order to reuse them later: when a request similar to an old one occurs, they reuse the value. They also use relaxed external procedures as heuristics to prune out part of the infeasible solutions before computing the complete external procedure.

Other approaches are also based on calls to external procedures, such as Ferrer-mestres et al. (2015) who worked on adapting a first-order planning language named Functional STRIPS by adding requests to external function (geometric tests for feasibility) as a component of a symbolic action. Guitton and Farges (2009a) and Guitton and Farges (2009b) also modify the symbolic action description, but they add geometric constraints to the action preconditions, which are passed to the geometric reasoner who use them to compute a new geometric state and then find a path from the previous geometric state to the new computed one. Gaschler et al. (2013a) and Gaschler et al. (2013b) also uses external call at symbolic level combined with a detailed symbolic state of the world –they are able to represent the state of a variable (known, unknown, incomplete or will be known at run time)– to compute feasible plans. Gaschler et al. (2015) extend this approach by adding specific geometric predicates to their actions, improving the search speed. Kaelbling and Lozano-Pérez (2011) use an aggressively hierarchical planner which stores the actions description primitives to compute and execute the actions. They use fluents to transform the geometric state to symbolic states and assess that the preconditions of the next actions holds or not. Kaelbling (2011) extend this approach by adding uncertainties: the planning is done in a hierarchical belief-space. The world is not necessarily known but is observable. Hence when performing an action, a previously unknown parameter might become known or partially known (looking inside a cupboard might end with knowing the position of a certain object or knowing that said object is not in the cupboard). Kaelbling and Lozano-Pérez (2013) extend even more the approach by adding domain models used as heuristics to guide and speed up the search in the robot belief-space.

Wolfe and Marthi (2010) and Wolfe et al. (2010) present an approach where they use high-level-action primitives as actions in an HTN planner. These actions can be refined to primitives such as navigate to somewhere, move arm to grasp or close gripper. Shivashankar et al. (2014) propose a formalism close to HTN but more directed to goal: Hierarchical Goal Networks (presented in Shivashankar et al. (2013)). They link this HGN planner with a geometric reasoner: every time an action must be tested they create the geometric state corresponding to the symbolic state and try to find a trajectory. If it fails, new geometric states are produced until the branching factor is reached (maximum number of allowed geometric states corresponding to a same symbolic state) in which case, the planner backtracks to the previous action. Once a trajectory is found, they compute its cost. If this cost is too high, the quality of the solution is considered not satisfying, then the computation of the corresponding plan is postponed and another is tried.

Some researchers also propose approaches including geometric backtracking. Alili et al. (2010) propose a combination of an HTN planner with a geometric reasoner, where a call to a geometric refinement is embedded in the symbolic actions preconditions. If a trajectory can not be found, the geometric layer tries to do a geometric backtrack: it keeps the symbolic plan and tries to reconsider the geometric decision made when choosing the parameters to find the trajectories of the symbolic actions. When all the actions have been reconsidered and if no satisfactory solution could be found then it tells the symbolic planner to propose a new plan, which triggers a symbolic backtrack. They also keep the symbolic state updated by computing facts after each geometric computation and handing them to the symbolic level. de Silva et al. (2013a) extend this work by adding literals protection: in the previous work, when changing the actions, the geometric backtracking does not check if the new proposed actions still produce the same outcome, which could mean that the symbolic plan is no longer consistent because the preconditions of some actions may no longer hold in the new geometric plan. Moreover the literals (which are facts passed to the symbolic level to assess some preconditions) are cached by the system for each task and protected when an action alternative is computed. Karlsson et al. (2012) depict a solution where, by using geometric backtracking with external calls and geometric literals (literals computed at geometric level and used at the symbolic level), they find feasible plans for a two-arm humanoid robot. Bidot et al. (2015) extend this approach, by first proposing a formal definition of the problem and then by adding geometric constraints able to guide the geometric backtracking in order to stress out the most interesting/constrained actions.

4.3.2 Geometric layer calls the symbolic layer

This category corresponds to the approaches where a geometric search is guided thanks to information from the symbolic layer. Zickler and Veloso (2009) perform their search in the geometric state space of the agents, where they compute, for each state, symbolic information enabling the search to be guided toward the goal. Choi and Amir (2009) propose to explore the model of the world with a motion planner algorithm (such as Rapidly-exploring Random Tree) and use the generated graph to automatically create feasible actions: if the motion generated by an edge (or a group of edges) of the graph, changes the state of an object, then it is considered as an action. Then a symbolic planner is used to find a plan using these actions. Nedunuri et al. (2014) base their work on an extended version of a manipulation graph –as in LaValle (2006)– which contains information about the robot base placement and arm placement to manipulate objects. They take a plan outline as a guide and then search through the possible sequences of actions available in the graph. Garrett et al. (2014a) and Garrett et al. (2014b) also use a graph capturing the possible manipulation actions in the environment and use the Fast-Forward task planner to find the best plan based on these actions. The graph is constructed by sampling the objects positions and computing some possible robots inverse kinematics, for each position, and then linking those configurations with trajectories. In Garrett et al. (2015) they change a bit the approach: they build a reachability graph from a simplified version of the problem in a backward fashion. From this graph they compute a heuristic that drives the (forward)

search to the final solution –the search itself is a persistent enforced hill climbing: they remember the minimum heuristic value and if a state with a lower value is reached then the queue of tasks is popped.

Plaku and Hager (2010) have a different approach where they sample the continuous space guided by the symbolic level, until reaching a state which satisfies the goal (this state is given to the geometric planner). In order to achieve this, they create a tree, and at each iteration of a loop, expand it by choosing the more relevant node (based on a utility function) and explore the space from there. Plaku (2012b) and Plaku (2012a) extend this approach by replacing the symbolic planner by an automata described using Linear Temporal Logic (LTL).

4.3.3 Sample in the compound state

In this last category, the search is carried out at the same time at geometric and symbolic levels, in a compound state. Actually one of the first works on combining symbolic and geometric planning is aSyMov in Cambon et al. (2003) and extended later in Cambon et al. (2004), Cambon et al. (2009), and Gravot et al. (2003), where the authors essentially propose a principled way to link the two planners thanks to a geometric level able to tackle the so-called "manipulation planning problem" first introduced in Alami et al. (1989), then in Latombe (1991) and that allows to explicitly take into account the topological changes occurring in the configuration space, when a robot grabs or releases an object. aSyMov provides a well-founded translation of pick and place actions (and similar actions) into "transit" and "transfer" motion planning requests even in multi-object and multi-robot contexts.

Hauser and Latombe (2009) consider that the robots can move inside a feasible space only, and can switch between "feasible spaces" through transitions: inside a "feasible space" the robot cannot change his contacts with the outside world (if he is moving an object for example) but can do it through a "transition space" (for example placing the object on a table). They create a Probabilistic Road Map (PRM, see Kavraki et al. (1996)) in each "feasible space" and aggregate them through milestones in the "transition spaces". Furthermore during the search they are able to begin the search in a direction, stop it, and postpone it (e.g. in case it is taking too long, so it explores other directions). Hauser (2010) extend this work by creating a symbolic language able to make requests to their previous system and by doing so, obtain a larger range of possible actions. This last paper enters in the sub-category of Section 4.3.1.3, as it makes requests to the geometric planner during the symbolic search. Barry et al. (2013) use a similar method to Hauser and Latombe (2009) but using a RRT algorithm instead of the PRM.

4.3.4 Synthesis and contributions

Table 4.7 shows the different works cited in this section organised by author, with some characteristics stressed out. Interestingly, Lagriffoul et al. (2013) argue that completely combining task and motion planning, as most of these approaches do, might not be efficient in every case: they are efficient

to solve geometrically complex problems but their performance might be less interesting than the classical approach when the problem is geometrically simple or trivial.

Our contribution extends the initial work done in de Silva et al. (2014) and are listed at the end of Table 4.7. It started with de Silva et al. (2013a) where they are using geometric backtrack, as mentioned before. But in the more recent years they switched to a system where the backtrack was done at symbolic level, and the geometric alternatives are represented as *instances* of actions in the HTN domain description. We presented the first formalism of the system in Lallement et al. (2014) and introduced the integration between the layers. We later extended it by adding the possibility to handle constraints and geometric knowledge in Gharbi et al. (2015), this is the focus of the rest of this chapter. In the most recent work we also change the backtrack mechanism, to prefer the most promising solutions, this work is presented in Chapter 5.

1 - Symbolic layer calls geometric layer
2 - Geometric layer calls the symbolic layer
3 - Sample in the compound state
4 - From a first symbolic plan find a geometrically feasible solution
5 - Ensure geometric feasibility while computing the symbolic plan
6 - Find a geometrically feasible plan among all symbolic plans
7 - Call to external procedures
8 - Compute geometric states from symbolic states
9 - Create symbolic knowledge from geometry
10 - Geometric alternatives
11 - Geometric backtracking
12 - Use constraints
13 - Account for uncertainties
14 - Using a graph covering the entire space

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Şucan and Kavraki (2011)	X					X								
Şucan and Kavraki (2012)	X					X							X	
Nedunuri et al. (2014)		X		X										X
Lagriffoul et al. (2012)	X			X						X	X	X		
Karlsson et al. (2012)	X				X				X	X	X			
Lagriffoul (2013)	X					X								
Lagriffoul et al. (2014)	X			X						X	X	X		
Bidot et al. (2015)	X				X		X		X	X	X	X		
Kaelbling (2011)	X				X				X				X	
Kaelbling and Lozano-Pérez (2011)	X				X				X					
Kaelbling and Lozano-Pérez (2013)	X				X				X				X	
Barry et al. (2013)			X					X		X				
Lozano-Pérez and Kaelbling (2013)	X			X								X		
Garrett et al. (2014a)		X							X	X				X
Garrett et al. (2014b)		X							X	X				X
Garrett et al. (2015)		X							X					X
Srivastava et al. (2013)	X			X					X					
Srivastava et al. (2014)	X			X				X	X	X	X			

1 - Symbolic layer calls geometric layer
2 - Geometric layer calls the symbolic layer
3 - Sample in the compound state
4 - From a first symbolic plan find a geometrically feasible solution
5 - Ensure geometric feasibility while computing the symbolic plan
6 - Find a geometrically feasible plan among all symbolic plans
7 - Call to external procedures
8 - Compute geometric states from symbolic states
9 - Create symbolic knowledge from geometry
10 - Geometric alternatives
11 - Geometric backtracking
12 - Use constraints
13 - Account for uncertainties
14 - Using a graph covering the entire space

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Caldiran et al. (2009a)	X			X					X					
Caldiran et al. (2009b)	X			X					X					
Haspalamutgil et al. (2010)	X			X					X					
Erdem et al. (2011)	X			X					X			X		
Havur et al. (2013)	X			X					X			X		
Gaschler et al. (2013a)	X				X		X						X	
Gaschler et al. (2013b)	X				X		X						X	
Gaschler et al. (2015)	X				X		X						X	
Dornhege et al. (2009)	X				X		X		X					
Eyerich et al. (2009a)	X				X		X		X					
Dornhege et al. (2010)	X				X		X		X					
Dornhege et al. (2012)	X				X		X		X					
Hertle et al. (2012)	X				X		X		X					
Dornhege et al. (2013)	X				X		X		X					
Plaku and Hager (2010)		X							X	X		X		X
Plaku (2012a)		X							X	X		X		X
Plaku (2012b)		X							X	X		X		X
Guitton and Farges (2009a)	X				X				X			X		
Guitton and Farges (2009b)	X				X				X			X		
Zickler and Veloso (2009)		X							X	X				
Choi and Amir (2009)		X							X	X				X

1 - Symbolic layer calls geometric layer
2 - Geometric layer calls the symbolic layer
3 - Sample in the compound state
4 - From a first symbolic plan find a geometrically feasible solution
5 - Ensure geometric feasibility while computing the symbolic plan
6 - Find a geometrically feasible plan among all symbolic plans
7 - Call to external procedures
8 - Compute geometric states from symbolic states
9 - Create symbolic knowledge from geometry
10 - Geometric alternatives
11 - Geometric backtracking
12 - Use constraints
13 - Account for uncertainties
14 - Using a graph covering the entire space

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Wolfe and Marthi (2010)	X				X		X							
Wolfe et al. (2010)	X				X		X							
Shivashankar et al. (2014)	X				X			X						
Dearden and Burbridge (2013)	X			X				X		X	X			
Ferrer-mestres et al. (2015)	X				X		X					X		
Hauser and Latombe (2009)			X					X		X				X
Hauser (2010)	X				X		X			X				
Cambon et al. (2003)			X					X	X					X
Gravot et al. (2003)			X					X	X					X
Cambon et al. (2004)			X					X	X					X
Cambon et al. (2009)			X					X	X					X
Alili et al. (2010)	X				X		X		X	X	X			
de Silva et al. (2013a)	X				X		X		X	X	X			
de Silva et al. (2013b)	X				X		X			X				
de Silva et al. (2014)	X				X		X			X				
Lallement et al. (2014)	X				X		X			X				
Gharbi et al. (2015)	X				X		X			X		X		

Table 4.7 – A synthetic reorganisation of the related work, sorted by characteristics and group of authors. Our contributions are in the last two rows of the table.

4.4 Formalism and algorithms

4.4.1 Geometric Task Planner

In our system, the geometric layer relies on the Geometric Task Planner (GTP) which is a framework that lies between the symbolic level and the geometric one. It is able to interpret high-level symbols and transform them into geometric trajectories, geometric entities, and so on. It can also transform the other way around extracting a set of symbolic literals from a world state $\mathscr{W}$.

Thanks to this ability to plan for geometric task, we can use GTP to assess the feasibility of a symbolic action in a given world state. The main benefit relies in the interpretation of the symbolic action into a geometric task: the symbolic layers does not have to handle any *geometric decision*, the geometric layers makes decisions on all free parameters. To plan for a symbolic action a into the current world state $\mathscr{W}$, GTP must decide on many parameter of the task: the grasping position or the gripper orientation on the grasp when picking an object, the object placement and rotation when placing it, and so on. Those decision would be hard or even impossible to consider by a symbolic planner. They are not equipped with the tools to process this kind of data and would be too limited or they would require a very complex model of the world that would create a combinatorial explosion when planning. When GTP has decided the parameters, the inverse kinematics are computed and finally an off-the-shelf motion planner finds the trajectories. So we have a mapping between a symbolic action and a geometric task.

The *projection* of a symbolic action into a geometric task and then into a set of trajectories is not the only abstraction GTP offers: it provides different levels of abstraction among the action it can interpret, for instance it can **Pick** (which is a sequence of reaching, grasping and lifting an object) and **Place**, but also **PickAndPlace**. This encapsulation allows for more clever and more efficient geometric computation. When a **Pick** is planned it may occur that some of the geometric decisions prevent some further **Place** (due to collisions for instance), which requires a backtrack to change the decisions and try to find another solution. On the other hand when a **PickAndPlace** is planned, all the geometric decisions are tested both in the initial and final configurations, and only then the trajectories are computed. Of course sometimes the robot may need to do some operations between the **Pick** and **Place** which justifies the need to keep all the level of abstractions, it allows a finer tuning.

After a geometric task is applied in a given world state, the configuration of the objects and robots will change creating a new world state. One of the features of GTP is to extract a set of symbolic literals from any world state. It computes geometric properties that can be **isOn**, **isIn**, **isNextTo** and so on, but it can also computes affordances (as defined in Gibson (1977)) such as **isVisible** and **isReachable**. Since those literals are symbolic but produced by the geometric layer we refer to them as "*shared literals*", and when they are requested after the projection of a symbolic action they represent its geometric effects.

4.4.2 Combining the planning modules

To achieve Symbolic and Geometric Planning (SGP) we are combining our planners such that HATP (symbolic layer) calls GTP (geometric layer) as defined by one of the categories presented above. The general principle is that as soon as HATP adds an action to the symbolic plan, it is projected, or instantiated, by GTP in the geometry. It also maintains a geometric plan that is the counterpart of the symbolic plan. This process is possible only thanks to the total order decomposition, because at any moment both the symbolic plan and its geometric counterpart allow to obtain the complete symbolic state and geometric world state allowing to project new actions in the geometry. It is important to point out that it is possible to have purely symbolic actions in the symbolic plan, which are actions that only affect the symbolic state and does not any impact on the geometric layer.

When an action is projected in the geometry, it is in fact encoded almost as an evaluable predicate in the preconditions. If the projection succeeds GTP computes and sends back the shared literals representing the outcome of the action on the geometry. Finally GTP also gives the length of the solution trajectory which is used as the action cost in the symbolic plan, hence ensuring that the action cost is accurate.

To be more precise the HATP language proposes a single "project" clause that does both the precondition test (for the projection) and that retrieves the effects. One of the parameters of the project clause allows to select the actual geometric action to use, and since this clause is a part of the description of a symbolic action it handles the mapping from a symbolic action to one of the geometric actions GTP can carry out. Then GTP generally decomposes those geometric actions into sets of interdependent geometric sub-problems resulting in the computation and selection of configurations such as the gripper position, the object placements, the trajectories. Those decompositions are given by hand by the geometric expert: for instance a **Pick** action corresponds to a trajectory to approach the gripper, then a small movement to reach the grasping position followed by the grasp, then a small escape trajectory moves a bit the object away from the table and finally the robot can retract to prepare the next trajectory.

The geometric actions need symbols to get the parameters of the task such as the object to move. The mapping from symbolic entities to geometric ones is done by hand (a system of aliases still offers to have different name but the mapping is static), where a geometric entity is a name associated to a mesh and a full configuration (position and value for the potential joints). This entities mapping is also used, the other way around, when GTP sends the literals back and another static mapping is used to transform the literals into entities attributes.

Listing 4.1 presents the *Pick* action where an agent picks an object if it is reachable (note the predicate where we test that the agent is in the list of agents that can reach the object). The use of project allows us to keep the code clean and the only form of control that we need is the number of backtrack alternatives created by the projection. Furthermore this clause also handles the geometric alternatives: when we backtrack to an action to ask for a new alternative, it actually requests an alternative instead which reuses the same initial world state as the original action (GTP knows this

world state thanks to the internal tree structure it uses to maintain the geometric counterpart of the symbolic plan).

```
1  action Pick(Agent R, Object O){
2      preconditions {
3          R.hasInHand == NULL;
4          R >> O.reachableBy;
5      };
6      projects {pickGTP{5}(R.getName(), O.getName())};
7      effects {
8          R.hasInHand = O;
9      };
10     cost{costGTP()};
11     duration{durationFn(1, 5)};
12 }
```

Listing 4.1 – *Pick* action with a geometric counterpart. Note the "project" clause that tells HATP which GTP task to call, the number right after the name tells the maximum number of alternatives to try. In cost clause we are using a function that retrieves the cost from GTP, it is possible to wrap this function call in more complex code to apply any desired mathematical function to this cost. The duration clause allows to specify the minimum and maximum time the function may last[4].

Indeed since some geometric choices are made in the process of projecting the action it is important to represent it in the symbolic layer with a backtrack point: we must be able to ask GTP for an alternative of the proposed projection if an action we want to add further in the plan can not be projected in the geometry or has invalid preconditions because of the geometric effects of a previous action. For instance if a robot is asked to place an object such that a human can pick it, we may need to ask for an alternative solution if the first proposed placement did not allow the human to actually reach the object (see Section 4.4.4 for more examples). As we mentioned it is possible to ask for alternatives, however if GTP fails to find a solution we consider that there is no (more) solution. Even though in theory there can be some, we consider it impossible in practice because of the implementation: for each choice GTP has to make it tries several hundred times, and tries different combinations between the configurations and the trajectories in order to find a suitable one. Hence if no solution could be found, we estimate that it exhausted the possible solutions it has in the context with the current parameters. This is a strong assumption but we never encountered, during our experiments, a case where GTP told us that there were no more solution while being wrong (or it would have required an impractical computation time in order of several minutes).

As the symbolic layer has no control over the geometric decisions we can only allow the symbolic layer to ask for alternative solutions of the projected actions. As of now the maximum number of alternatives is fixed by hand, and are associated to a backtrack point of type(BP) = $geometric$ and task(BP) = a. In Listing 4.1, for instance, five alternative choices are allowed which means that GTP may change the grasp position on the object, the gripper orientation and so on up to five times.

[4]The duration function is used only to compute an estimated total duration for the plan, and GTP can not compute it for now. In future work we want to retrieve it from the trajectory and also use it more in the solution plan to allow better monitoring: the supervisor could know the time limits and using the causal link could better behave when encountering an error.

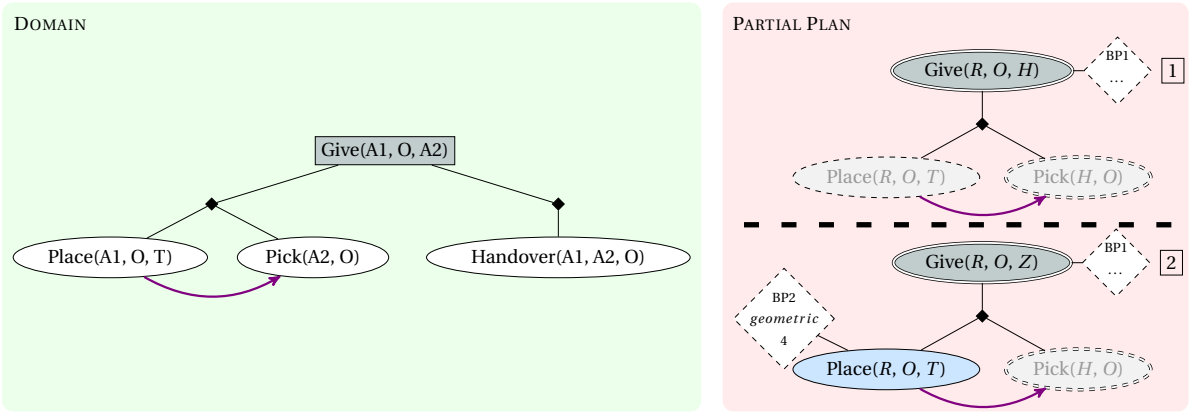


Figure 4.2 – Illustration of how HATP adds an action with a geometric counterpart. A backtrack point, BP1, was already present to remember the method choice, now a new one, BP2, has been created to store the maximum number of alternatives minus one (four in this example) geometric alternatives for the *Place* action.

Figure 4.2 shows HATP internal structure when an action with a geometric counterpart is added. An illustration of the backtrack process can be seen in Section 4.4.4.2.

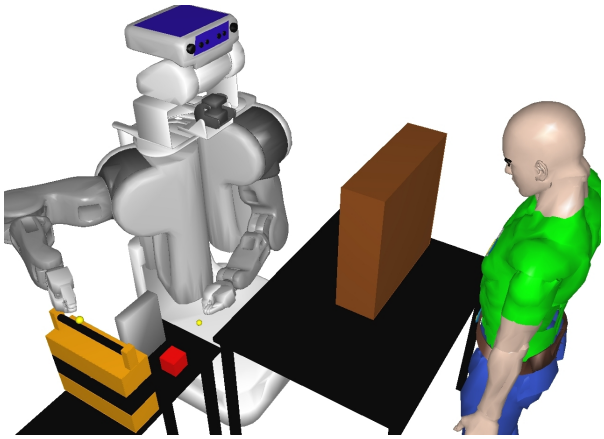
4.4.3 The ramification problem

The ramification problem comes from the impossibility to specify all the effects of certain actions: there are implicit effects. It is a problem already encountered in the symbolic planning community, but all the more important when combining symbolic and geometric planning: the placement of an object can change a lot of things and its environment, for instance it could prevent any further trajectories by blocking the path to other objects. It is even more obvious when computing affordances (visibility, reachability and so on), for instance if the robot moves an object it can easily occlude another object. To tackle this problem GTP uses the shared literals to send the effects of the actions to HATP. By taking into account those literals it is possible to understand the effects and side effects of the actions and reason about them.

Figure 4.3 shows an example where a robot has to place three objects such that a human can reach them at the same time. Depending on the position of the objects they may prevent the human to reach them, furthermore sometimes when placing a new object it "breaks" the reachability to an object that was accessible before.

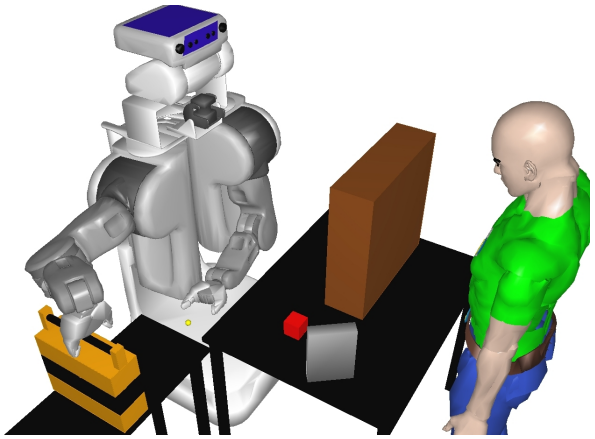
To handle undesired side effects, it is necessary to test the corresponding predicates in the domain. It can be done either in the preconditions of one of the next tasks or if no task depends on those literals then it may be necessary to add an empty task that serves as a geometric goal. It will only have preconditions and no body: its aim is to trigger a backtrack (it can be a method with no subtask or an action with no effect). As soon as an undesired effect is detected, HATP can backtrack to ask for alternatives on the geometric actions and could expect there exists a geometric solution

RedCube.reachableBy={Robot};
GreyBook.reachableBy={Robot};
OrangeBox.reachableBy={Robot};



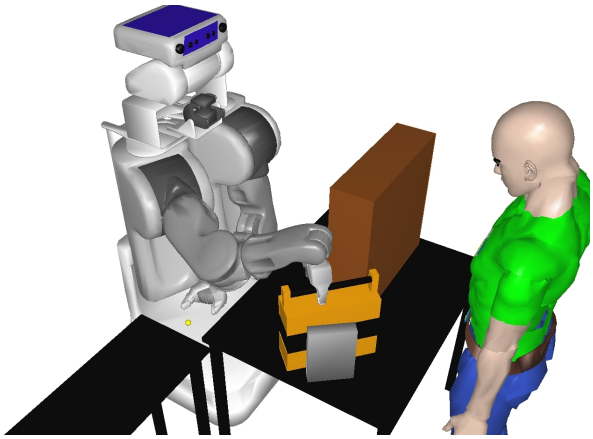
(a)

*RedCube.reachableBy={Robot, **Human**};*
*GreyBook.reachableBy={Robot, **Human**};*
OrangeBox.reachableBy={Robot};



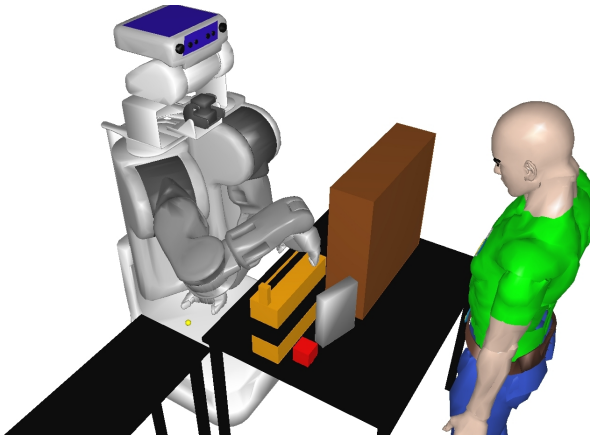
(b)

RedCube.reachableBy={Robot}, ~~Human~~;
GreyBook.reachableBy={Robot, Human};
*OrangeBox.reachableBy={Robot, **Human**};*



(c)

RedCube.reachableBy={Robot, Human};
GreyBook.reachableBy={Robot, Human};
OrangeBox.reachableBy={Robot, Human};



(d)

Figure 4.3 – Illustration of the ramification problem, the robot has to place the three objects such that they are all accessible to the human at the end. The initial state is represented in 4.3(a). In 4.3(b) the robot has already placed two objects, and if it places the third as in 4.3(c) we can see that the red cube that was previously reachable becomes inaccessible by the orange box. After some backtracks (moving all the objects) the robot can find a solution like in 4.3(d) where all the objects are accessible.

where the undesired effects do not occur. An illustration of this mechanism in action is shown in Section 4.4.4.2.

This mechanism was one of the main limitations of our algorithm, shared with many other approaches: it may require to backtrack a lot before finding a suitable solution. Let's assume we detect an undesired effect produced by task a_i (after a_{i-1} tasks) and then many other actions $a_{i+1} \dots a_{i+k}$ (k new actions) were projected. Then we will have to ask for all the alternatives of all the k new tasks before finally asking for a new alternative for a_i. Furthermore the number of alternative to test, along with number of backtrack to trigger, grows exponentially: if there are n alternatives for each task in $a_{i+1} \dots a_{i+k}$, then we have to backtrack $n-1$ times for a_{i+k}, then $n-1$ times for a_{i+k-1} followed again by the n times for the new a_{i+k} since the world state changed, and so on. In total we will have to project n^k actions before asking for each new alternative of a_i which obviously requires a lot of time. Later in this thesis we propose two solutions to address this limitation.

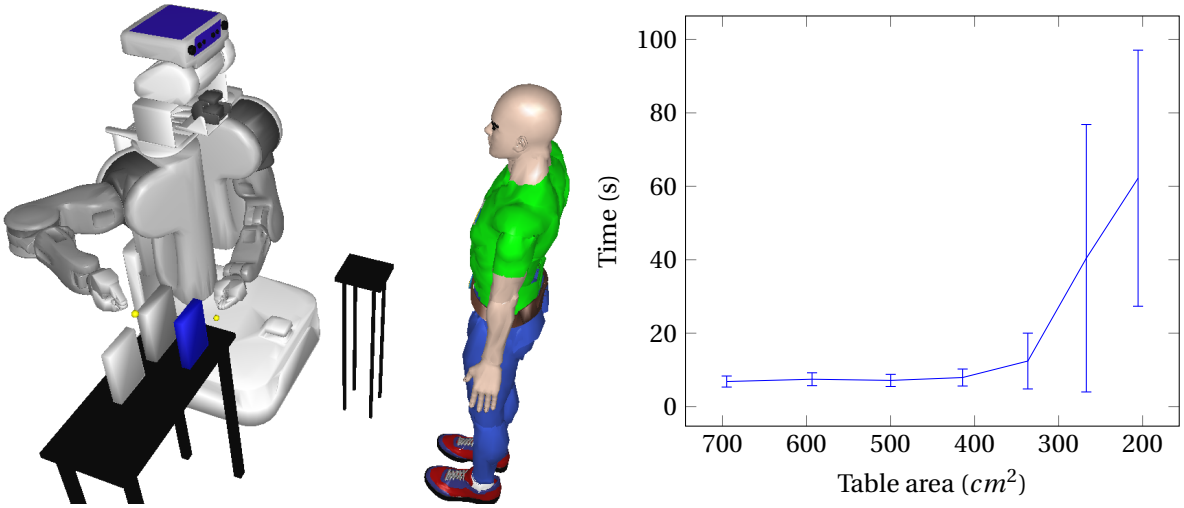
The ramification problem is only partially tackled due to the discrete set of shared literals the system is able to compute: if a shared literal does not exist, the problem will obviously not be tackled.

4.4.4 Example of domains

This section demonstrates our system in a variety of domains. More advanced ones are shown in other parts of the thesis to demonstrate specific features and improvements.

4.4.4.1 Place on a small surface, difficult for SGP

The goal of this example is just to demonstrate how difficult it is for SGP to find solutions even for simple problems, because of the geometry. The robot has to place three books on a small table, in front of a human such that all the books fit on the table altogether. Figure 4.4(a) shows the environment where the small table in between the human and the robot is the target table, its size is the variable of the domain as shown in Figure 4.4(b). We portray this problem with a small table which size changes, but it could be a bigger table and the area then would represent the surface of the table the robot can reach (the rest being too far or blocked by an obstacle). This example showcases how the computation time changes as the problem becomes more difficult, there are less and less acceptable positions for the books. We can also observe an increasing standard deviation: we have no guarantee over the geometric planning time, a solution can be found very early (thanks to "luck") or the search can go on for long time. The plot only shows the computation time, we could have displayed the number of requests from HATP to GTP but its profile is exactly the same as the computation time one. In other word there is a direct link, in this particular domain, between the computation time and the number of requests, we do not have extra computation time. We could have expected the computation time to grow faster than the number of requests: the motion plans could be harder to find, however GTP starts by trying to find a final position for the objects before computing the trajectories. So the real problem comes from the decisions made over the position of the objects, usually the first ones can be placed but then we backtrack several times to reorganise them.



(a) The scenario, with the small table

(b) The mean computation time (and standard deviation)

Figure 4.4 – In the scenario, on the left, the robot must place three books on the small table, which size is a variable of the problem. On the right we display the computation time for different table sizes (its area expressed in cm^2). The table on the left has an area of $15 \times 17.8 = 267 cm^2$ and from a point to the next on the curve, the width and length are both increased by $2cm$, proving how much a small change can have a big impact on the computation time of SGP problems. The same problem occurs with a bigger table which surface is not completely accessible to the robot (if there are obstacles, or some parts are out of the robot reach).

4.4.4.2 Place reachable example, handle the ramification

This section studies more in-depth the example depicted in Figure 4.3. As aforementioned, the goal is for the robot to place the three objects such that they are all accessible to the human at the same time. The initial state is represented in 4.3(a) and one of the possible final world state that satisfies the goal in 4.3(b).

We propose two variants on the domain description to show how the abstraction given by GTP can improve the computation time: (1) the first variant use two actions: *Pick* and *Place*, (2) the other variant relies on a different "place" action: *PlaceReachable*. This new action is similar to a normal *Place*, the only difference is that GTP computes a final position for the object such that it is reachable to an agent (given as a parameter of the action). The first variant is represented in Figure 4.5, the second variant only changes the *Place* action. The *Validate* abstract task controls that the current state is valid (i.e. all objects already placed are reachable) and has no subtasks, it is only a precondition with no decomposition.

The top-level task *GiveCollection* which calls three *Move* tasks, one for each object, does not choose the order to use when placing the objects. In order to highlight the improvement provided by the reachability computation we fix the order to be the worst case: place the objects from the smallest

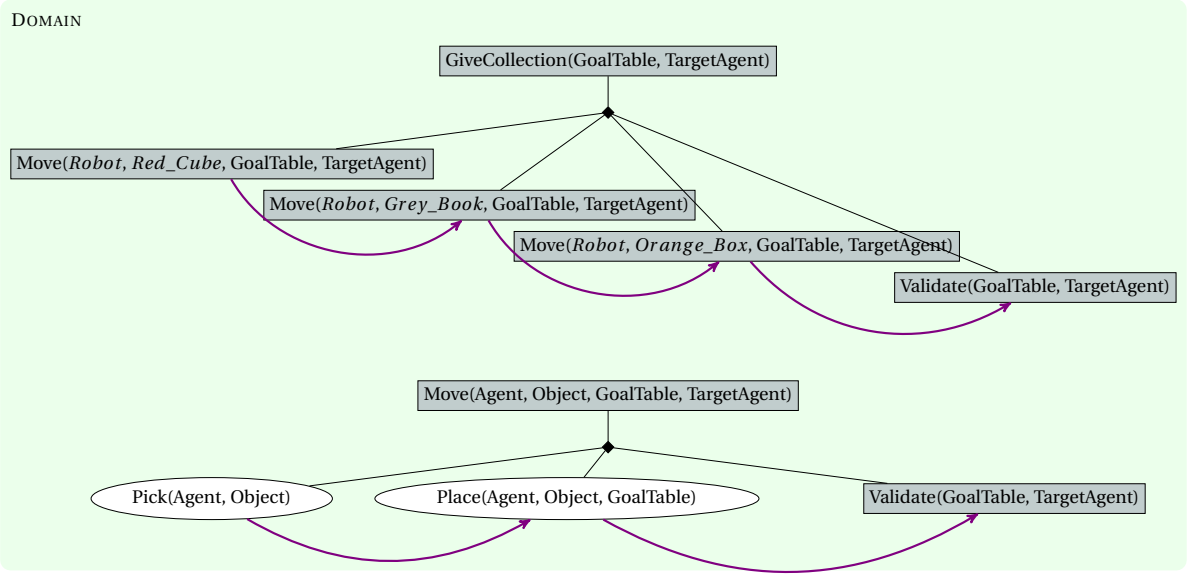
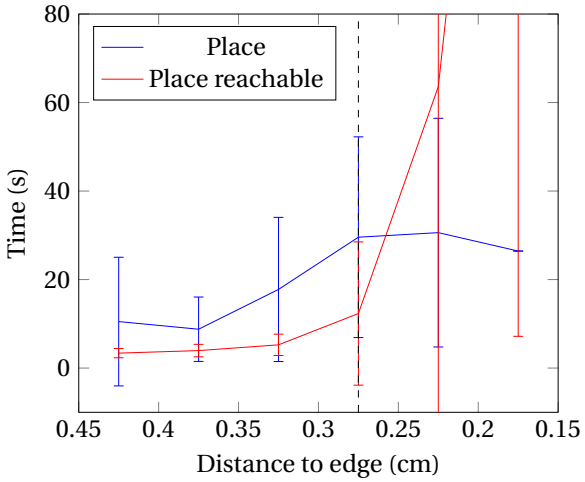


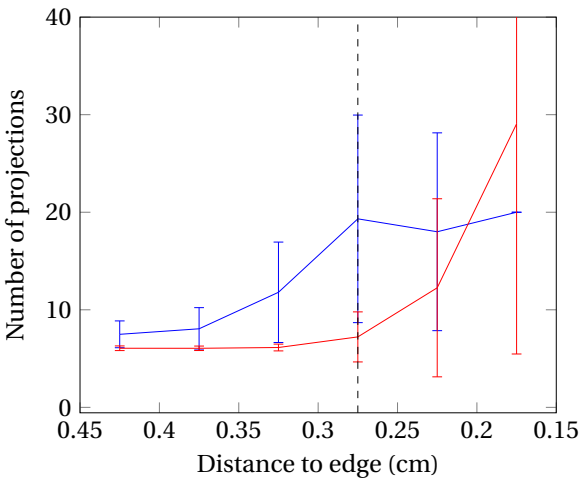
Figure 4.5 – Domain description for the two variants of the "place reachable" scenario: the first variant is represented in the figure, the second variant has exactly the same domain description only the *Place(Agent, Object, GoalTable)* action is replaced by *PlaceReachable(Agent, Object, GoalTable, TargetAgent)*.

to the biggest. This order is the worst case because when the biggest object is placed last it has more chance of occluding the reachability to the others. We can of course let HATP choose the order to handle the objects but when the order is the best then the two variants are equivalent, thus using the worst-case scenario helps to isolate on the computation-time improvement.

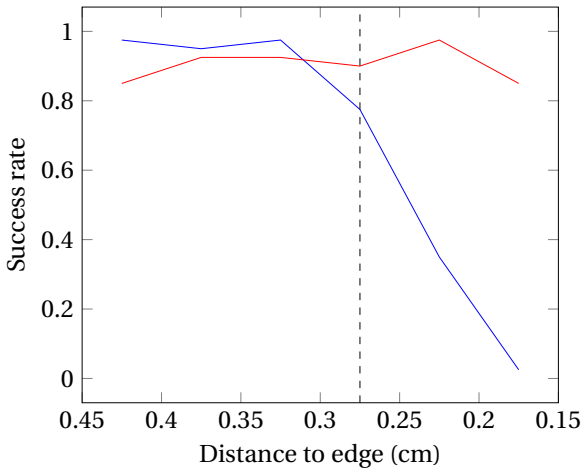
Figure 4.6 depicts the results of both variants (*Place* and *PlaceReachable*) where the distance between the brown obstacle and the edge of the table changes. The goal is to illustrates the impact of the complexity and the gain of using a higher level task. The dashed vertical line highlights the two different "phases". First, when the problem is not very complex (big distance to the edge, left of the plots) the *Place* and *PlaceReachable* variants achieve a similar success rate but we can clearly see that the *PlaceReachable* variant is much more efficient and deviates a lot less from a run to another. In the second phase we see that as the problem becomes difficult the success rate of the *Place* variant falls rapidly reaching almost zero while *PlaceReachable* maintains a good success rate. It is very important then to understand the difference in computation time and number of projected actions. The *Place* variant seems to be efficient while in fact the values –for both the time and number of tasks– correspond to "lucky" runs, the very few that actually led to success. We did not plot the time spent searching for unfruitful solutions. On the other hand *PlaceReachable* has reasonable number of projections but its computation time increases much faster: the trajectories become more and more complex to compute because of the obstacle in the way.



(a) The mean computation time (and standard deviation)



(b) The mean number of projected actions (and standard deviation)



(c) The success rate

Figure 4.6 – Results for the place reachable domain with two variants: *Place* versus *PlaceReachable*.

Figure 4.7 shows an instance of a search where the goal state is not reached and some backtracks are necessary. It is important to note that the reachability of the objects are tested (at symbolic level) either at the very end or after a new object has been placed and a new *Move* method is about to be decomposed. Which means that when a new *Move* is decomposed all the previously placed objects were reachable at the beginning although there is no guarantee that it remains true at the end. Even in the second scenario GTP ensures only the reachability of the new one, not the others. This is the main reason for backtracks, a newly placed object broke the reachability of another which is detected by the next *Validate* task.

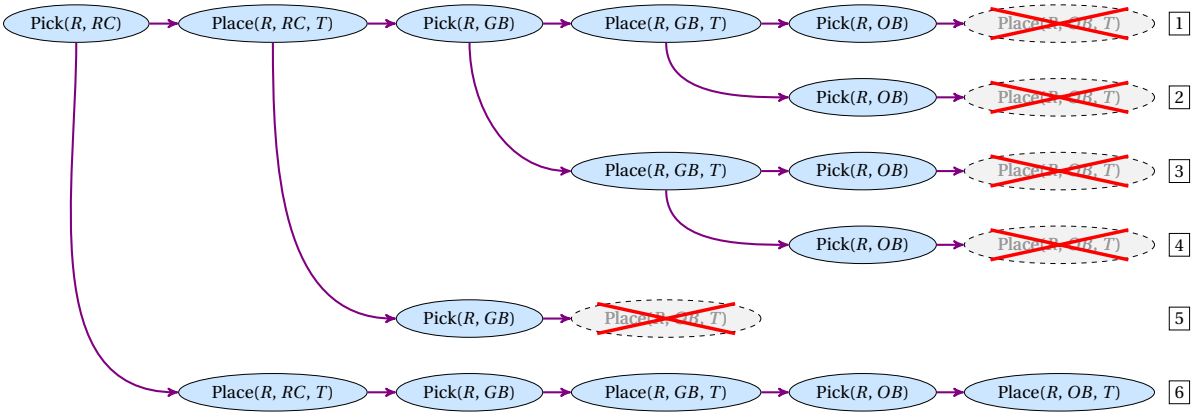


Figure 4.7 – Demonstration of the backtrack mechanism in the "place reachable" scenario. The first variant is used in this figure: HATP controls that no object previously placed is now unreachable. In the first four attempts the last *Place* fails[5], the objects were not all reachable. In the fifth case the occlusion occurred earlier so there is no need to plan further since it can only fail. Finally the sixth attempt leads to a valid plan. This illustrates the ramification: the system takes in account the side effects of the actions. Without the shared literals sent from GTP to HATP, the symbolic level would have no way to know that an action has broken the reachability of other objects.

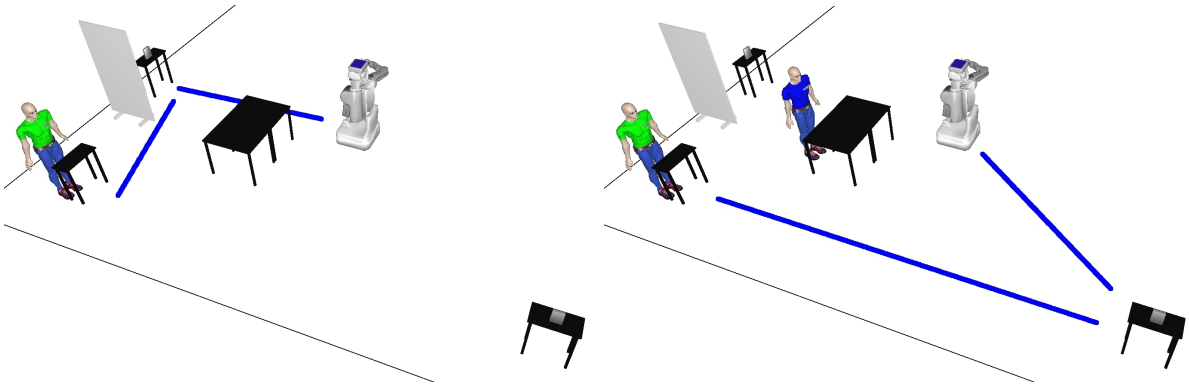
This scenario demonstrates how our system handles the ramification problem. Thanks to GTP sending the shared literals to HATP, the symbolic layer has a better understanding of the effects for the actions and it can reason about it. Thus if an action breaks the reachability of an object previously placed, the symbolic level can backtrack and ask the geometric layer for alternatives. We count on the fact that GTP will be able to find a configuration of the objects such that the goal state is reached, and in order to increase our chances of getting such state we must abide to a rapidly increasing computation time.

4.4.4.3 Use geometric cost to improve social interaction

To exhibit a socially acceptable behaviour the robots have to reason about the location of the humans, and GTP can reason about it (see Mainprice et al. (2011)). It values differently the paths depending on the position of the human relatively to the trajectories. Indeed it takes into account socially acceptable trajectories and comfort distances: too close or behind a human. Hence when following the path if the robot goes too close (or behind) a human the trajectory is considered to not be socially acceptable and is evaluated with a big penalty. Since HATP uses the cost returned by GTP as the cost of the actions with geometric counterpart, it is then possible to reason about those social costs because the penalties are retrieved and are used to search for the better solutions.

[5]When a *Place* is crossed in this figure, it exceptionally means that the following *Validate* task failed. Moreover the *Validate* task is never represented in Figure 4.7 for compactness.

In the example depicted in Figure 4.8 a human wants the robot to bring him a book for which we have several copies. So the robot has the choice between two books and must bring either one of them to a human. This domain takes place in a workspace where a human co-worker can be working so the robot must adapt its behaviour. In 4.8(a) the co-worker is away so the robot can use the shortest path, but in 4.8(b) the robot must follow the longer path to stay away from the co-worker. This choice is possible because the two books are symbolically similar, which HATP can reason about, and the path cost depends on the geometric cost only GTP can compute: the integration between the two layers allows to solve this problem while having consideration for the etiquette.



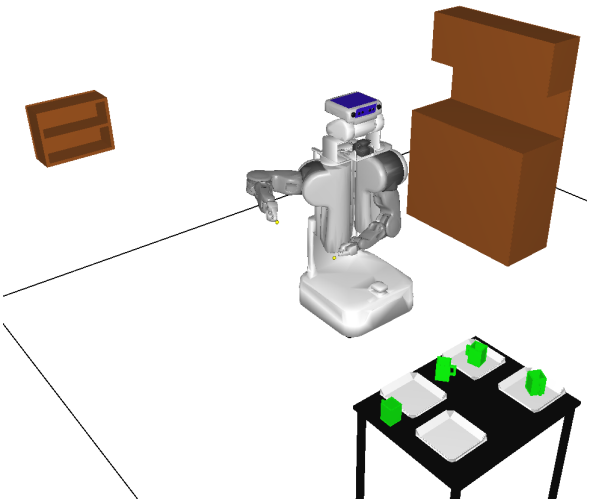
(a) Environment without the worker

(b) Environment with the worker

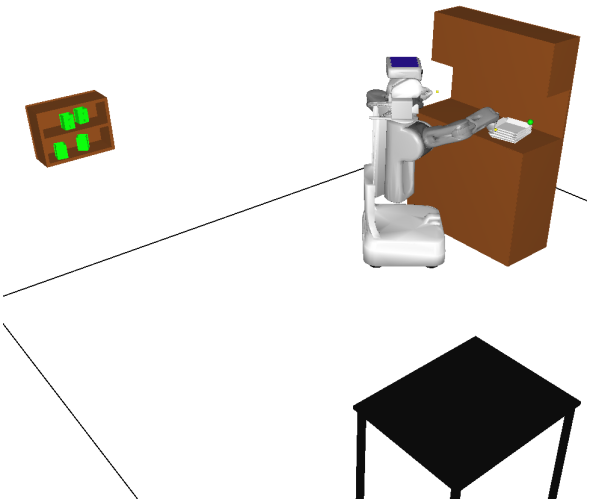
Figure 4.8 – Illustration of the use of the geometric cost. The robot must pick on of the two grey books (they serve the same purpose) and bring it to the counter. In 4.8(a) the robot can go to the object with the smallest total distance necessary, on the other hand in 4.8(b) the robot must take a longer path to avoid going behind the other human: GTP puts a penalty on paths going behind humans (or too close to them).

4.4.4.4 Complex domain

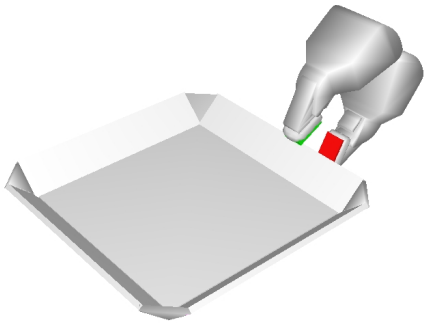
This last scenario consists in a robot that must store mugs and plates to their respective storages, as in Figure 4.9. The mugs must be stored in a cupboard, while the plates must be placed on a closet (or can be stacked). There are many problems to solve when planning in this scenario, the first is that the robot is forbidden from moving the plates when there is something on top. It can either store the object that is on top first or it can move that object away in order to get the plate. For the mugs the difficulty comes from the grasp position, the robot is unable to pick them from the top and place them in the cupboard because its arm will be in collision. So the robot must grasp from the side, but it still is a decision that GTP must make (to prove that the system is capable of handling difficult situations). When storing the mugs in the cupboard the robot must choose which shelf to use, there are two, it can use either of them but if there are already many mugs on a shelf it may fail to place the new one on this shelf as well. Finally the last choice the system must make occurs when the robot



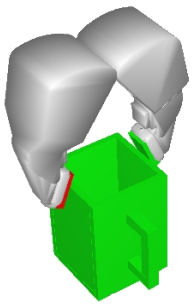
(a) Initial state.



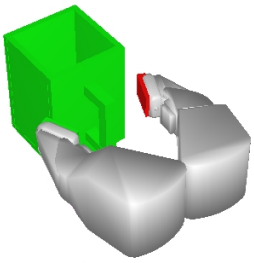
(b) Example of final state.



(c) One of the grasp for a plate, the gripper can be anywhere around the edge of the plate with a similar grasp as the one depicted.



(d) An example of top grasp on a mug, this grasp prevents the robot from placing the mug in the cupboard, a side grasp must be used.



(e) This type of grasp allows the robot to place the mug in the cupboard. All the sides of the mug can be used, not just the handle.

Figure 4.9 – Domain where the robot must store the mugs and plates, to their respective storage: the mugs go to the cupboard (left piece of furniture) and the plates to closet (right piece of furniture). The robot has many options: it can move the mugs away from the plates to make them available, on the closet the plates can be stored side by side (very tight fit) or can be stacked. But it also has constraints: the mugs must be held by the side in order for the arm to be able to place the mug in the cupboard.

stores a plate on the closet, it can either put it on the closet or stack it on another plate in order to reduce the number of plates directly on the closet (hence reducing the complexity of placing a new plate).

We ran this domain fifty times and the average computation time to find the first feasible solution is around 30 seconds. This first plan consists in the robot carrying all the mugs first and then the plates. However it is possible to find other plans (roughly equivalent in term of cost) where the robot first brings the two uncovered plates, then the mugs blocking the other plates, then the plates that

are no longer blocked and finally the remaining mugs. Almost any actions order is valid as long as all the constraints are respected.

A video of this example running can been seen there: https://homepages.laas.fr/rlalleme/en/multimedia.html.

4.5 Enhance the search with knowledge

In this section we present some improvements on the integration between the layers in order to have a more efficient search both at symbolic and geometric level. Those enhancements were the preliminary steps to improve the integration that led us to propose more advanced techniques that are the topic of the next chapter.

4.5.1 Constraints from the symbolic module

The first improvement aims at reducing the geometric search space as well as drive the search with symbolic information. When GTP projects a symbolic action it is possible to add a set of constraints: a set of symbolic literals that GTP must enforce in the end world state of the action. An example of such constraint is the *PlaceReachable* action we presented before, GTP must ensure the reachability after projecting a *Place* action. The power of this constraint mechanism lies in the way GTP resolves them.

HTN encoding is very suited to such constraints mechanism because it is designed to offer a global view on the problem and sub-problems (the decomposition). This hindsight allows the domain expert to know what are the important "goals" to reach in each task hence she/he can add the appropriate constraints, moreover they can be placed where they matter thanks to the hierarchical nature of the HTN domain.

When GTP is asked to project an action it starts by determining the final configuration of the objects to fulfill the action, then it computes the inverse kinematics to find the final configuration of the agents and only then it computes the motion from the initial world state to the final world state. To be efficient GTP takes the constraints into account as soon as possible: for instance if there is a reachability constraints on an object, then it will be taken into account in the first planning step. Since all the constraints are taken into account before the motion planning phase if they happen to be infeasible (conflicting between themselves or with the current environment) GTP will be able to tell very soon that the action can not be planned without wasting time.

To prove its efficiency we implemented it in the domain depicted in Figure 4.10. The goal of the robot is simply to put the yellow box on the table such that all the objects are reachable to the human (the yellow box and the two small red cubes). The complexity of this domain lies in the small window that the robot must use to pass the yellow box. Indeed this causes the GTP planning time to be huge because this kind of problem is hard to solve for RRT planners. The domain encoding is kept simple because most of the details are given by hand, there is no symbolic choice. About the

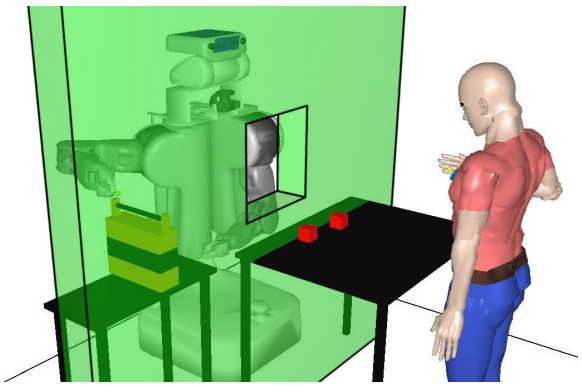


Figure 4.10 – Domain to test the *Place* with constraints. The robot must *PlaceReachable* the yellow box on the table passing through the small window. At the end the three objects, the yellow box and the two red cubes, must all be reachable to the human. The complexity comes from the RRT computation that struggles to find solutions in such a reduced search space.

tasks, we have an abstract task *Move* that *Pick*s then *PlaceReachable* and finish by *Check*ing that all the objects are reachable to the human. We have a variant of this domain where the *PlaceReachable* action is replaced with an action, *PlaceConstraints*, that places the object reachable but also that adds constraints to GTP: all other objects must stay reachable.

In this second variant, since the constraints are enforced early, the planner finds a position for the box that is suitable before even trying the motion planning phase. Whereas when constraints are not used only the symbolic layer can detect the broken reachability and trigger a backtrack and ask for an alternative. We can see in Table 4.8 that thanks to the constraints the system is seemingly faster. It is interesting to note that the difference in the number of projected actions is not as big as the difference in computation time: this illustrates the difficulty GTP has to plan such trajectories.

40 runs (time in s)	Place Reachable	Place Constraints
Mean time (stdev)	59.154 (43.542)	2.425 (0.712)
Number projected actions (stdev)	44.376 (44.318)	2 (0)

Table 4.8 – Measure of the improvement due to the constraints over *PlaceReachable* alone. Both type of actions are placing the object reachable, but only the constraint version ensures that the placement preserves the reachability to the objects already there. The complexity comes from the trajectory computation, passing through a small window is hard for RRT which makes the computation time very long.

We think that the constraints can be used in two cases: when one wants to protect facts from previous actions (as in the example above), or to prepare some facts for later actions. For instance the *PlaceReachable* action uses such constraint to ensure that the object we place will be reachable later. We tried to run the *PlaceConstraints* action in the domain where the robot has to place several

objects, see Figure 4.3 to see how much the system would benefit of the constraints to protect all the previous reachabilities. However the space around the obstacle is mostly free which means that the problem really comes from the choice of the position rather than the trajectory computation.

For now the constraints are encoded in the domain as in Listing 4.2, but in future work we wish to extract them automatically. A pre-planning step could do a limited decomposition of the plan to determine what are the causal links between the tasks and then extract some constraints to prepare the important facts. Additionally in Section 4.5.3 we present a solution to explicitly state the tasks goals, we could use those goals to determine what are the facts to enforce along their decomposition and add the appropriate constraints.

```
1 action ConstrainedPlaceReachable(Agent R, Object O, Surface To, Agent A){
2     preconditions {
3         R.hasInHand == O;
4         FORALL(Object C, {C.isOn == To; A >> C.reachableBy;},
5             {addConstraint(C.getName(), "isReachableBy.isReachableByIK", A.getName(), true)==true;});
6     };
7     projects {placeReachableGTP{5}(R.getName(), O.getName(), To.getName(), A.getName())};
8     ...
9 }
```

Listing 4.2 – In this code snippet we can see how the constraints are added to protect the reachability of all the objects already on the destination surface.

4.5.2 Exploiting the geometry as heuristic

In order to get knowledge from the geometry we have developed a way to ask GTP to plan virtual actions, those actions do not involve the motion planning and are not kept by GTP in its plan structure. Furthermore those virtual actions corresponds to some actions GTP can plan (e.g. **Place**). The reason for such actions is to be able to test the final configuration of a virtual action to determine beforehand if a decomposition involving the actual action has chances of success. If the virtual action can not be planned then there will be no solution for the actual action (it will not be able to find a configuration since they were already tried). The idea behind the virtual actions is to encode “common sense” into the domain description. Usually when we want to achieve an action, for instance carrying a load, we visualise the object at its final location to see if the action is possible. With the virtual actions the domain expert can imitate that behaviour and tell the system when to assess the possibility of doing an action before trying it, hence saving time.

Even though the virtual action could be planned we have no guarantee of success for the action it represents. For instance a placement for an object could be found, so the virtual action would tell us that it is possible to place the object while the actual trajectory could be impossible due to obstacle. The virtual actions do not take into account the motion so they may miss collisions on the path or any similar problem.

To further extend this idea we also created virtual objects that are meant to represent objects or set of objects when trying out virtual actions. The goal of virtual objects is to allow to test if a bigger object could fit then it would mean that the actual object will surely fit (quite like a bounding box).

The main usage of a virtual object, though, is to capture a set of objects: we create a virtual object big enough to hold all the smaller objects of the set (and where the robot could place them without collision). Hence when we project a virtual action with this virtual object, if it can be placed then we are sure that all the objects of the set can be placed as well.

This heuristic-like system ensures the success of the next action if the virtual action succeeded, whether the test used a real object or a virtual object, only under the condition that the world state did not change too much. We do not provide any mechanism to ensure that, because we consider those heuristics as tools for the domain description, we expect the domain expert to use them correctly. On the other hand, if the virtual action failed there are two cases: if it was invoked with the actual object then it means the object can not be placed and there is no need to compute the motion plan since a placement could not be found. However if the virtual action used a virtual object then we have not form of guarantee since the object is bigger than the actual object, maybe a tight fit would allow the object to be placed.

HATP uses those virtual actions and objects to prune out methods or to reduce the possible values for a variable binding. Listing 4.3 depicts the usage of a virtual object (the big box in Figure 4.11) and a virtual action: *VirtualPlace*. The *GiveCollection* abstract task has two possible decompositions: if the virtual object can be placed then we try to place all the objects directly. On the other hand if the virtual place can not be done we first remove objects from the table until it is possible to place the virtual object and only then we place the collection of objects.

```
1  method GiveCollection(Surface To, Todo Todo) {
2      goal {FORALL(Object O, {O >> Todo.objects;}, {O.isOn == To;});};
3      {
4          preconditions { //If the virtual object can be placed
5              EXIST(VirtualObject V, {}, {virtualPlace(V.getName(), To.getName()) == true;});
6          };
7          subtasks {
8              //Directly place the objects
9              1: PutObjects(To, Todo);
10         };
11     }
12     {
13         preconditions { //If the virtual can not be placed
14         };
15         subtasks {
16             //First empty the table
17             1: CleanTable(To, Todo);
18
19             //Then give the collection
20             2: PutObjects(To, Todo)>1;
21         };
22     }
23 }
```

Listing 4.3 – The *GiveCollection* abstract task is the root of the HTN tree and decides whether to directly place the object on the table or to first move any object before placing the collection of objects. We can see on Line reflst:virtualObject:Exist how to use the virtual object and the virtual action: in this scenario we have only one virtual object so the EXIST clause will select it, then it is tried in its final position thanks to the *VirtualPlace* virtual action accessed through the evaluable predicate: virtualPlace. *The Todo parameter holds the list of objects to place, and we explain the goal clause in the next section.*

In Figure 4.11 this form of heuristic helps the system decide on the decomposition to use. The system must place the three books on the table in front of the human, but there may be some objects already on this table, in that case the human or the robot may have to move out the objects. The heuristic we use consists in (virtually) placing a virtual object that is big enough to contain the three books. Thus if this virtual object can be placed the system directly chooses to place the books, but if the virtual object can not fit then the system removes the objects before placing the books. This is very different from the classical approach where the robot would have to first try several times to place the three books before giving up, removing the objects and trying again to place the books. However this approach is only a heuristic and may lead to sub-optimal solutions: in 4.11(f) the virtual object can not be placed without collision so the system will remove the objects before placing the three books however they could have been placed even without removing the objects.

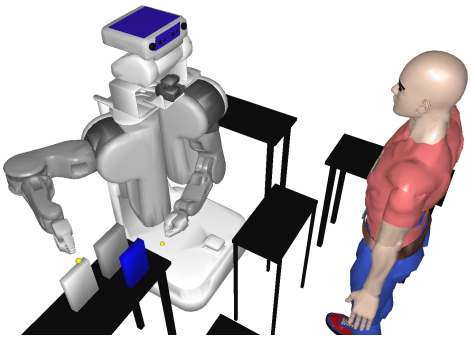
In Figure 4.11 we show three of the eleven scenarios in which we demonstrated out virtual object and virtual *Place* action mechanism. In the other scenarios we have used the same tables setup but changed the number of objects on the target table in the initial state: no object, a single small or big object, several objects and so on. We aimed at proposing a variety such that the we represent both the cases where the heuristic is better, is equivalent and is worse than the planner alone. The overall results can been seen in Table 4.9, we see a general speed-up and a reduction of tasks tried. In three of the eleven scenarios (not necessarily the ones from the figure), the use of virtual *Place* would be sub-optimal however such solutions always consist in removing the objects and then placing the three books which in most of the scenarios prove to be almost as good as the best solution since it clears up the table hence reducing the complexity of placing the new objects.

	Without virtual *Place*	With virtual *Place*
Mean time (stdev)	191.2 (8.6)	21.7 (1.45)
Nb projected actions	162.6	25

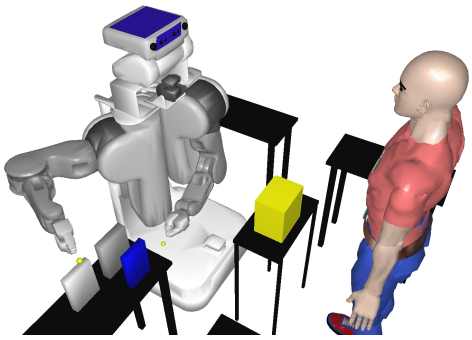
Table 4.9 – Results of running the system with and without the virtual place on a variety of scenarios. The speed-up is obvious, although there are three scenarios where the use of the virtual *Place* would be sub-optimal. The time wasted moving objects away than did not prevent the completion of the goal and still helped reducing the difficulty of the next actions.

4.5.3 High-level goals

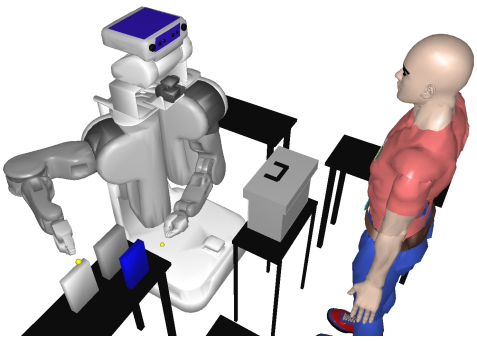
The last improvement is focused on the symbolic layer: we add an explicit goal to the abstract tasks. The aim of this construct is to be able to state the goal attached to the task such that after the task decomposition we can assess that its objective is reached. If it failed we can trigger a backtrack to either use another decomposition or to change the choices made in this decomposition in order to get a suitable solution that fulfil the goal. This mechanism allows to tackle side effects of actions: we can partially specify what facts we want or not in the final symbolic state. Thus we are able to detect



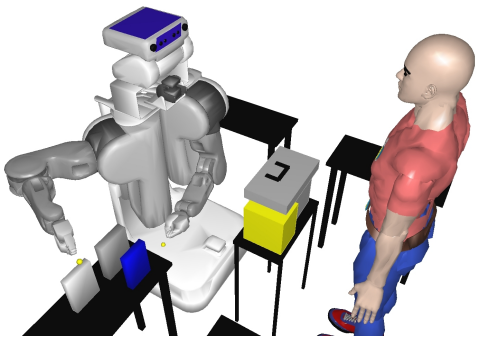
(a) Scenario 1, initial state



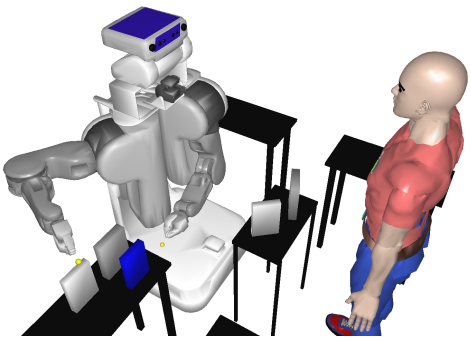
(b) Scenario 1, virtual object can be placed



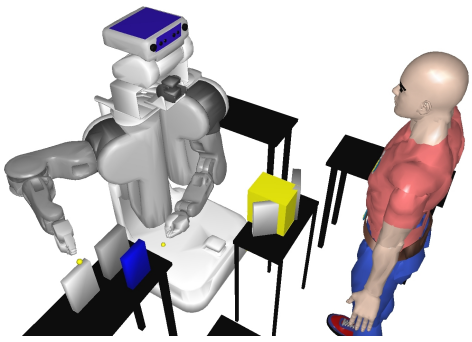
(c) Scenario 2, initial state



(d) Scenario 2, virtual object cannot be placed



(e) Scenario 3, initial state



(f) Scenario 3, virtual object cannot be placed

Figure 4.11 – Illustration of the virtual *Place* action. The robot must place the three books (at its right) on the table in front of the human, which may require to remove objects already present on the table. To determine if objects must be removed, the system tries to randomly *Place* a virtual object (yellow box in this example) at many location on the table. If it could place the yellow box, as in 4.11(b), then it proceeds to place the books. However if it fails to place the virtual object it will remove the objects from the table before placing the books. In the scenario 3, however the books could have been placed without removing the two objects already present.

any deviation: an action did not produce the expected outcome for instance (producing an extra unwanted fact, or missing some of its effects).

We replace the definition of an HATP method:

$$m_{hatp} = \langle \text{name}(m_{hatp}), \text{emtpy}(m_{hatp}), \{d_0 = \langle \text{precond}(d_0), \text{subtasks}(d_0), \text{constr}(d_0)\rangle, \ldots\}\rangle$$

The new definition is:

$$m_{hatp} = \langle \text{name}(m_{hatp}), \text{goal}(m_{hatp}), \{d_0 = \langle \text{precond}(d_0), \text{subtasks}(d_0), \text{constr}(d_0)\rangle, \ldots\}\rangle$$

We have changed the empty function with goal, the former was used to represent the empty methods preventing decomposition or stopping the recursion. The latter is a bit more general it keeps the same meaning but also serves at checking after the method decomposition that its goal is reached, similar to contract programming. The empty clause already represented the "goal" of the abstract task because when all its predicates are true then we considered that the task was already achieved (its goal was reached) and we were not decomposing the task. The new usage introduced by the goal function consists in controlling that the goal has been reached when the decomposition is finished. This approach allows to tackle the ramification, indeed when the domain description is sound this has no effect when planning purely symbolic actions but if the actions have a geometric counterpart it proves to be efficient to promptly detect a problem such as a "broken" predicate.

In HTN the abstract task have an implicit goal, there name for instance gives an idea of this goal. Making it explicit encourages the domain expert to formalise the task goal as a set of predicates that must hold after the task decomposition. In the domain of the "place reachable" we even represented this goal in the form of an empty task (see Figure 4.5). we can prove that such empty task (abstract of operator) are equivalent, it is just a cleaner way to obtain this result. Furthermore this formalisation is also an opening to future work on heuristics: it could be used to determine the heuristic cost of decomposing the task in the current state.

4.6 Conclusion

In this section we started by presenting the state of the art and proposed an organisation of the different papers according to different criteria. We then introduced the work carried out by our predecessor: they first combined task and motion planning using geometric backtrack, then created the premises of our current system.

To reason on both the symbolic and geometric layer at the same time we developed the Geometric Task Planner (GTP) that abstracts the motion planner so it can interact with the symbolic layer, HATP. We introduced a new type of action that have a geometric counterpart: a motion must be found for the action to be feasible and considered valid by the symbolic level that can then add it to the symbolic plan. Consequently the symbolic planning process changed, when such an action is encountered HATP asks the geometric layer if it is feasible. However this request uses a symbolic definition for the

action and it cannot be understood as is by a motion planner, hence GTP transforms the symbolic action into a set of geometric tasks and has to take decisions on some of their parameters: for instance a position for an object such that its final position corresponds to the action. Since the geometric layer makes decisions we can ask for several alternatives for a single symbolic action. To represent and manipulate it HATP creates a backtrack point on those actions with a geometric counterpart.

Since the environment changes when an action is applied in the geometry its effects must be taken into account at the symbolic level to remain consistent (ramification problem), so GTP computes a set of symbolic literals that are sent to HATP: the *shared literals*. They allow us to reason on the effect of the actions and we demonstrated in some scenarios that they allow to adapt the plan to the action outcome.

Finally we introduced some improvements to tighten this integration of symbolic and geometric planning: (1) how to drive the geometric search with constraints from the symbolic layer, (2) how to extract information from the geometry and use it as a symbolic heuristic and (3) how to detect an invalid decomposition as soon as possible by working with task goals.

As mentioned in the introduction of this chapter, the work presented here has been realised jointly with Mamoun Gharbi. Most of the contributions implied a very tight cooperation to both define the problems, the interfaces and implement them in the different tools. This process was mainly iterative going back and forth between the two tools. Mamoun Gharbi handled most of the the work conducted on the geometric layer thanks to his background, while we focused on the symbolic layer for the same reason.

Although we have an integration that allows to plan complex solution involving geometric and symbolic planning, taking into account the side effects of the actions, we have a limitation: the planning time. Too much planning time is wasted backtracking on points that are not relevant, moreover the number of backtrack alternatives grows exponentially. The next chapter focuses on two approaches to solve this issue.

CHAPTER 5

IMPROVING THE SGP EFFICIENCY

This chapter presents two enhancements to the HTN search algorithm aimed at reducing the computation time. The first improvement addresses the problem of the time wasted on "useless" backtracks. Because of the depth first mechanism and the exponential number of alternatives (mainly geometric) in the search tree the symbolic planner wastes time exploring all the backtracks in a chronological fashion. The first improvement proposes a mechanism to rate the backtrack points according to criteria that mix symbolic and geometric information.

This first method proves to be efficient and offers a significant speed-up, however it is only used when the system backtracks. The second enhancement is a proposal where the depth-first is replaced by a heuristic search, similar to A*.

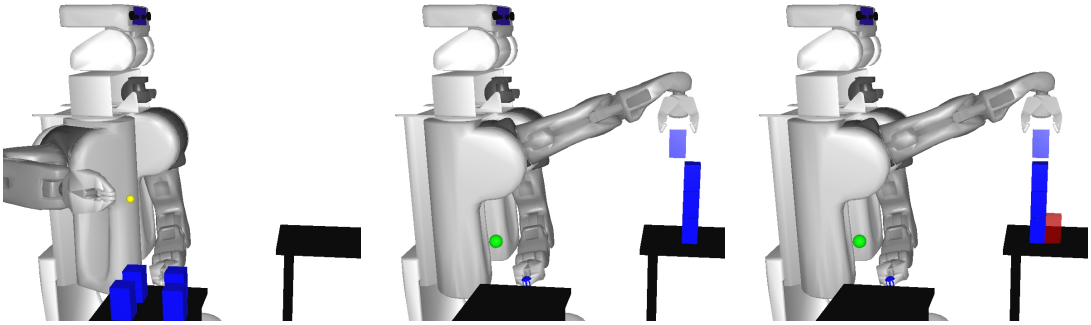
5.1 Toward an informed backtrack mechanism

As said before, the planner can explore the different alternatives that the geometric layer proposes, however since the sum of all the possible alternatives –to backtrack to– grows exponentially with the number of actions in the plan, to use a chronological backtrack proves to be slow in most cases.

5.1.1 Problem description

A motivating example helps to see the need to improve the backtrack mechanism, let us consider the example illustrated in figure 5.1: the robot needs to stack the four objects, but if the stack is too far, the robot cannot stack the last object since it cannot reach the position. In this example, when the pile is too far, it will first try all the alternatives for the last *Pick* (since there are different grasping positions) before trying the *Stack* action which has no alternative in out scenario (it would be possible to have several rotations of the object but in our case we consider it to be fixed). But then it will

succeed to find alternatives for the third object placement so it will try to *Pick* and *Stack* the fourth object. Only after exhausting all alternatives for the third object placement it will try to change its *Pick*. This problem will again be faced with the second object and finally the first object placement will be reconsidered. It is clear that in this scenario the number of backtracks is exponential in the number of objects, while only the placement of the very first object actually has an impact on the chance of success, basically it determines the position of the stack. Although the problem is trivial HATP has no way to understand this simplicity, we need a mean to provide it with the geometric knowledge telling it only the first *Place* action matters.



(a) (b) (c)

Figure 5.1 – Motivating example for a new backtrack mechanism. In this example the robot must create a stack with the four objects, but because of the final size of the pile and because of the position of the table if the robots starts the pile too far from itself it will fail to finish the pile. (The red cube in the third figure, shows the position of the stack from the second figure.) With the default strategy to change the position of the first element of the pile requires a great number of backtracks.

We remind all the information on the backtrack points in the table below and introduce the *backtrack alternative* noted ba:

BP	A backtrack point
type(BP)	The type of the backtrack point, can be $decompo$, $bind$, $order$ or $geometric$
task(BP)	The task that created this backtrack point
ba	A backtrack alternative from BP, noted $ba \in BP$
status(ba)	The status of the alternative, can be $unexplored$, $success$ or $failed$. If ba has been explored then the status tells if a plan has been found or if a backtrack was necessary
$\mathscr{L}_{BP}$	The set of all the "open" backtrack points, for BP to be open means that it contains one unexplored ba: $\exists ba \in BP$|status(ba) = $unexplored$
$\forall ba \in \mathscr{L}_{BP}$	All backtrack alternatives that are still unexplored[1]

[1] To keep the notation simple, $\forall ba \in \mathscr{L}_{BP}$ means all the unexplored backtrack alternatives of all the opened backtrack points. It is equivalent to $\forall ba \in BP$|status(ba) = $unexplored, \forall BP \in \mathscr{L}_{BP}$.

Our idea consists in rating the different backtrack points and their alternatives in order to drive the search toward the most promising points. In order to ease the rating process, we propose to create a combination of some criteria rating each backtrack alternative according to a simple rule. The criteria can be of different nature: based on purely symbolic reasoning, or geometric reasoning or mixing both types. Obviously only the backtrack alternatives still not explored are rated since the others will never be visited again. Indeed the backtrack alternatives already visited led either to a failure or success and are now "closed". The criteria can be seen as functions: $\omega : ba \mapsto [0,1]$. They return a value between 0 and 1 for convenience since it is easier to picture and to combine; the higher the score the more promising the backtrack alternative is. Depending on the backtrack reason, some criteria will be enabled while others will not, which allows to fine-tune the influence of each criterion.

It is important, at this point, to note that the use of our backtrack ranking approach does not impact the completeness. Indeed we are just changing the order in which the backtrack alternatives are visited, we are still exploring all of them (a backtrack alternative valued to 0 means it will be explored last, not that it should be skipped).

5.1.2 Criteria mechanism

This section presents some of the criteria we have implemented, they can still be optimized and may not be the best criteria possible but they already offer a good speed-up and are rather intuitive, most of the time corresponding to common-sense knowledge. Some of the criteria presented require a knowledge coming from a geometric expert since they imply to understand how the geometric planner works. This knowledge integration helps finding the solution which corresponds to the intent behind HTN planning.

5.1.2.1 Place-causality criterion

The first criterion is action specific and is related to the computation around *Place* actions causalities. Two actions may have a strong bound, the effect of the first action may be mandatory for the second to be applicable, whether it is for its symbolic precondition or for the way the (geometric) world state $\mathscr{W}$ changed. It is important to differentiate those causal links from the precedence links "$\prec$" because the causal links are not explicitly stated in the domain $\mathscr{D}$. Moreover the symbolic layer does not possess the mean to understand the part about the geometric causalities. Hence some geometric knowledge is needed to model the interdependence. The best example of the causal links is between a *Place* action and some previous actions: the *Pick* action corresponding to this *Place* action (the grasp can prevent the placement) or previous *Place* actions that were done on the same surface (the objects placed before can prevent the placement of the new object).

So we propose the following three sub-criteria that are explained in the following sections and which are related to: (1) detect the *Pick* corresponding to the failed *Place*, (2) the previous *Place* actions occurring on the same surface and (3) the symbolic choice of the surface where to do the current *Place*. When combined they give a criterion efficient in scenarios such as the one depicted in

Figure 5.2. The goal is to place the three objects on any of the two small tables, if an object is placed in the middle of a table it will prevent any further *Place* on the same table. Each table can hold the three objects but it requires to correctly place the objects. The improvement given by this criterion can be seen in the Section 5.1.4.1.

***Pick* corresponding to the *Place* action** When the robot grasps an object to pick it, the geometric planner has to choose the position where to place the gripper on the object. For many objects there are several ways to handle them, so this criterion tries to tackle this diversity. Indeed a given grasp can prevent the robot from placing the object, for instance a cup must not be held by its bottom when it must be stacked.

This criterion values the backtrack alternatives corresponding to the *Pick* of the object we are trying to place:

$$\forall ba \in \mathscr{L}_{BP} \text{ such that type}(BP) = geo,$$

$$\omega(ba) = \begin{cases} 1 & \text{if } ba \in BP \text{ which corresponds to the last} \\ & \text{pick action with the same agent and object} \\ 0 & \text{otherwise} \end{cases}$$

***Place* at the same location** If one or several objects were placed at the same location (destination) they may prevent the current *Place* action because the objects are not well organized and take too much space (not their volume but their organization on the target location). If we try again to choose a new position for some of those objects we may find a configuration such that the new object can be placed.

So the criterion increases the probability to search for an alternative to those *Place* actions:

$$\forall ba \in \mathscr{L}_{BP} \text{ such that type}(BP) = geo,$$

$$\omega(ba) = \begin{cases} 1 & \text{if } ba \in BP \text{ which corresponds to any place} \\ & \text{action at the same location} \\ 0 & \text{otherwise} \end{cases}$$

Symbolic choice of location for this *Place* Finally the last sub-criterion reasons about the location where the current *Place* action failed, maybe it is too cluttered and another location would be more suitable.

In this case the criterion cares about the backtrack alternatives of variable binding ($ba \in \mathscr{L}_{BP}|\text{type}(BP) = bind$) leading to this location, setting all the alternatives corresponding to all possible values for the variable to 1 (all other points will receive a 0).

5.1.2.2 Stack-causality criterion

The idea of this criterion is inspired from the first criterion, it works on the causality reasoning for a *Stack* action. Such action derives from a *Place* action by the fact that corresponds to placing an object on top of an object places before with the constraint of being centred (relatively to the support object). The idea is to have an action with less degrees of freedom to speed-up the computation.

Some causalities on a *Stack* action are inherited from the *Place* action (a *Stack* action is a *Place* action in essence): it depends on the corresponding *Pick*, on the previous *Stack* actions (their angle to be more precise) and the symbolic choice of the pile of objects the new one is stacked on. Additionally it has a very strong dependency on the *Place* action of the first object of the pile. Indeed the first object is "placed" on the table (*Place* action), while all other objects are "stacked" (*Stack* action) on top of it. Although a *Place* action has many degrees of freedom (position and orientation), a *Stack* action on the other hand does not have any since even the angle is fixed (this action can also be used for assemblies, and a StackGTP action is used in the project clause). For instance if the pile has been started too far from the robot it may succeed to stack some of the objects but will fail to stack the last objects because of the reach of its arm. So moving the first object placed (and bringing the pile closer) will have a big impact on the rest of the plan.

Hence we are reusing the sub-criterion of Pick-causality, the symbolic choice and the previous *Stack* actions (previous actions with the same type). But we also need a new sub-criterion which will set to 1 the *Place* action that started this stack.

5.1.2.3 Causality on broken predicates

This criterion addresses symbolic reasoning about the (geometric) actions effects[2] and unmet pre-conditions. The symbolic planner maintains the list of facts an action produces (shared literals) when added to the plan, and this list is usually used to create causal links between actions (e.g. of different agents). When a task is not applicable because some of its precondition predicates are not met then the purpose of this criterion is to reason about the task responsible for the failure. It will increase the value of the backtrack alternatives corresponding to the (geometric) action that prevents a predicate to hold (the criterion solves one flaw at a time). This criterion helps solve the ramification problem, indeed when an action has side effects that change a fact which was needed by another action it is probably more relevant to backtrack to the geometric choices done for this action.

5.1.2.4 Placement-collision criterion

This criterion is inspired from Bidot et al. (2015) where the objects preventing a trajectory are used to determine the backtrack alternatives. The principle is to detect the objects that may have prevented an action to be carried out in the geometry, and use this list of objects to favour the backtrack

[2] In a sound domain, only the action with a geometric counterpart may have side-effects. Indeed the geometric action can have effects that change the visibility (hiding or revealing an object) that are not part of the main intended effects (i.e. in a move action).

alternatives corresponding to actions that placed those objects. However sometimes we encounter collisions with a movable object from the environment that was not moved in the current plan, then this object is not considered (this criterion is used to select a backtrack point).

To compute the list of collisions it is possible to store, for each object, the number of times the motion planner points were in collision (during the search). Then thanks to its algorithm GTP can tell us when it failed due to collision. Therefore combining it with the list of collisions we can find the object with the most collisions and try to backtrack to its *Place* action. When GTP fails to instantiate an action with geometric counterpart, it tells HATP the reason of failure. When it fails due to collisions (other types of failure can: inverse kinematic, no grasp, and so on) it tells HATP the number of collision for each moveable object. HATP then uses it to retrieve the *Place* actions that might have created this problem.

5.1.2.5 Planning-difficulty criterion

The last criterion presented in this thesis is just a proposal and is focused on the action "difficulty" in the sense of how hard they are to (geometrically) plan for. The idea is to measure this difficulty either through time spent planning or number of nodes opened during the search (to be more accurate it will be divided by the length of the solution trajectory, to take into account the relative difficulty already brought by the distance to cover).

The goal of this criterion is to explore first the "cheap" branches, the aim is not to stick to solution that are expensive to explore (constrained environments). When an error occurs in a costly branch, we switch to a branch that may be cheaper thus exploring it faster. It is important to note that a different version of this criterion could be useful in domains we know have a high complexity: hence we would backtrack on part that are actually problematic first in order to know sooner if the solution could be found at all.

This criterion is not implement yet, but the principle would be to value with a "default" value when alternatives are created. This value could be either given by hand (by the geometric expert for instance) or be the mean value of all the previous runs. It would serve as a neutral value: higher value would mean that the action is difficult, while smaller value would mean that the action is easy to plan (empty space for instance). When an action with geometric counterpart is added to the plan we measure the geometric planning time, and store it as the value for all the other alternatives of the same action. It can happen that an action is difficult to plan because of the (random) decisions made by GTP such as the placement and that an alternative of this action would be a lot easier to plan just by changing this placement. This means that sometimes we will over-evaluate or under-evaluate the difficulty of alternatives for an action because of the first we planned. So to reduce this risk the planning time representing all the unexplored alternatives will be the mean value of all the alternatives already explored for the same task.

5.1.3 Combination and final selection

We just presented some criteria but there exist many others possible. Since they are not mutually exclusive, and even can enrich one another we need a way to combine them. Moreover some criteria may be useless depending on the reason why the backtrack is needed.

Our system allows the domain expert to provide a description of the criteria to use. It is possible to specify activation conditions such as the name of the task that failed. Furthermore a combination of criteria can be given and is expressed as a weighted sum:

$$\text{rate}(ba) = \sum_{i=0}^{n} \alpha_i \times \omega_i(ba)$$

where ω_i represents a given criterion if it is enabled. When no criterion is enabled, the depth-first strategy is used as a fallback. For now this combination is hard-coded and must be re-written for each domain. In future work we want to develop a small expressive language to complete the domain description with the combination of its backtrack criteria: enabling only the relevant ones and changing the weighs according to the context.

For instance, if a *Place* action is not applicable in the current world state $\mathcal{W}$ the system will enable the Place-causality criterion alone, but it could add the complexity criterion to mainly select backtrack alternatives which are easier to plan (if the place locations are cluttered) to prevent a planning time too big. But none of those criteria would be triggered if a backtrack is necessary because the first plan is found and the system needs to find a better one.

When the combination of criteria is computed, all the backtrack alternatives left are rated, the system then needs to select only one to backtrack to. First it finds what is the highest value, and picks all the alternatives with that value (L_{ba}). After this first selection some alternatives can belong to the same backtrack point, they represent different values for the same choice (i.e. all the values a variable could be bound to), then the system retrieves all the backtrack points that have at least one alternative from the first list (L_{bp}).

$$\forall ba_i | \text{rate}(ba_i) = max, L_{ba} = L_{ba} \cup ba_i$$
$$\forall BP_j | \exists ba_i \in BP_j \wedge ba_i \in L_{ba}, L_{bp} = L_{bp} \cup BP_j$$

To backtrack the planner randomly picks a backtrack point BP_α from L_{bp}, then in this backtrack point it will randomly pick an alternative ba_β among the one from the first list: $ba_\beta \in BP_\alpha \wedge ba_\beta \in L_{ba}$.

This two-stage selection is necessary to prevent some types of alternatives to be over represented. For instance when a variable is bound it may have only few values (let us say 2) while a *Place* action can have many more alternatives (let us say 20). The backtrack tree would contain only one backtrack alternative for the variable binding, while it would have 19 for the *Place* action so it would be less probable to select the binding even though it is as significant as the choice over the *Pick* alternatives. So to first *Pick* the backtrack point and only then to choose the backtrack alternative allows us to even the odds.

5.1.4 Experiments and results

In order to properly compare the basic depth-first mechanism and the improved backtrack-point selection, we are using the same domains to run both cases. All the scenarios were run 50 times to encompass the widely random nature of the geometric layer (both for placements and trajectories).

In this section we show the result of three criteria, the two last criteria presented (placement collision and planning difficulty) were not implemented and serve as prospects.

5.1.4.1 Placing objects with few possible choices

In this first scenario, Figure 5.2, the robot must place objects on two small tables, the choice over the table to use is left free. Each table can holds the three objects but they must be placed very precisely. The problem that the robot usually faces is that it can not place the third object because the two previous occupy too much space on the chosen table.

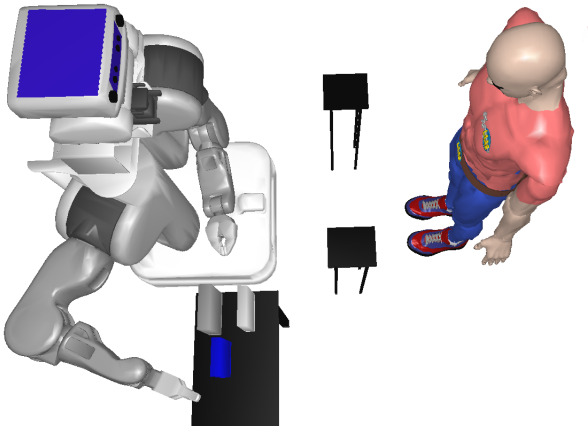


Figure 5.2 – A scenario where the three objects must be placed on the two small tables. If placed correctly all can be placed on a single table, but this requires a precise organisation. A traditional HTN may backtrack a lot before finding the solution, while our solutions focuses on the alternatives for the places (position on the table, or the table used), and it finds the solution significantly faster.

With the “place-causality” criterion the robot will focus on trying a different table or move the objects on the first table until they leave enough room for the last one (it may also test the *Pick* but it occurs less often than in the general case). On the other hand the classical backtrack will try many times the *Pick* actions, then try alternatives for the previous *Place* before even considering changing the table. Table 5.1 shows the results for 50 runs. It presents the number of backtracks in regard with the planning time and number of calls to the geometric layer. It compares the results obtained with the depth first backtrack against the one obtained with the Place-causality criterion.

Analysing this first table shows that our system requires significantly-less backtrack (reaching several order of magnitude in the worst case). Obviously this decrease can also be seen on the number of GTP calls. The speed-up is even more significant considering that we are allowing only 3

	Number of backtracks			Planning time (s)			Geometric calls		
50 runs	Min	Mean (stdev)	Max	Min	Mean (stdev)	Max	Min	Mean (stdev)	Max
Depth first	21	72.5 (282.5)	2041	15.5	64.6 (240.1)	1737.5	36	115.2 (400.9)	2880
Place criterion	4	23.0 (13.3)	63	9.8	36.9 (18.9)	88.9	12	52.0 (28.5)	135

Table 5.1 – The results of the Place-causality criterion against the classical HTN approach (depth-first search) when placing the three books on the two tables as in Figure 5.2

alternatives on *Pick* actions and 10 on *Place* actions (those number were determined to be efficient in most cases).

5.1.4.2 Stack example

This scenario, shown in Figure 5.1, requires the robot to place the four objects in a stack (the order is insignificant so it is given by hand). However because of the position of the table, the robot can easily fail to stack the four objects because the last one will be out of reach. Any position for the first object (the stack base) any further than 5 cm from the table edge prevents the robot to build the stack, and the furthest the stack is the sooner the robot fails. Additionally we are allowing the robot to grasp only from the top.

Table 5.2 shows that number of backtracks is significantly decreased. Again we are allowing 3 alternatives on *Pick* actions and 10 on *Place* actions but we do not allow any alternatives on the *Stack* action (the rotation is not relevant in the scenario). What makes the classical approach so slow, even though there are only few alternatives is the chronological backtrack: it will spend time trying to change the *Pick* actions and then try the *Stack* actions.

	Number of backtracks			Planning time (s)			Geometric calls		
50 runs	Min	Mean (stdev)	Max	Min	Pred. crit.	Max	Min	Mean (stdev)	Max
Depth first	10	40.6 (24.9)	134	7.1	26.9 (14.9)	82.2	33	123.7 (74.0)	398
Stack criterion	2	3.2 (1.7)	9	3.1	5.7 (2.1)	11.4	10	19.8 (7.8)	51

Table 5.2 – The results of the Stack-causality criterion against the classical HTN approach (depth-first search) on the stack scenario as in Figure 5.1.

5.1.5 Extending the use of the criteria

As we illustrated in the experiments there is a significant speed-up in the planning process when using our new backtrack mechanism, however our process is limited since it works only when a backtrack is needed. An interesting extension would be to use the computation done by the geometric layer to drive the decomposition. The most straight-forward way to do it would be to use it when selecting a method or binding a value.

Indeed we could have a set of functions that could sort the entities according to geometric information. A first step in this direction was presented in Section 4.5.2 where geometric functions are used to pre-assess the feasibility of a given actions that will appear in the decomposition, thus filtering the "invalid" decompositions. It could be extended by using it also to sort the different decompositions or values for a variable binding in a task. An example to illustrate this idea: a robot has to fetch an object among many equivalent ones, then to choose the object to pick we could use the euclidean distance to determine the closest. Combining this sorting with the backtrack could prove to be efficient.

The next section explores a way to go further: replace the depth-first search with a heuristic-driven search like A*. This section is just a opening and has not been implemented during the time of the thesis, however it presents in detail the formalism that could lead to an implementation.

5.2 Replace the HTN depth-first search

This paper proposes a formalism for a new search algorithm for HATP. Indeed the depth-first algorithm is not efficient enough in the context of Combined Task and Motion Planning (CTAMP). Most of the ideas presented here could be applied to purely-symbolic domains but we will focus here on the CTAMP problem.

5.2.1 Motivation and expectation

When planning in CTAMP with HATP we stop, most of the time, at the first solution. To find the best solution would require to explore the complete decomposition tree and try all the alternatives both symbolic and geometric (also give the geometric layer more time to find a better solution) which would require very long computation time or a post-process to optimise the solution. However we consider that finding the first plan that is geometrically feasible is enough, because the symbolic planner should be driven toward the best solution thanks to the domain encoding so we expect the first solution to be already good. Indeed we use the domain description to drive the symbolic search toward a "good enough" solution, for instance by providing the methods in their order of quality and using ordering functions to bind variables, then when the solution is built we use the geometry to filter out the infeasible plans. Although in many problems instances the time spent backtracking is significant which in some cases might even be too high to be considered: the depth first approach can lead to unreasonable computation time.

When we proposed the improved backtrack mechanism we intended at reducing the computation time by focusing on promising solutions. The problem is that this selection occurs only when a backtrack is triggered. However the approach is relevant to drive the search using the domain encoding that is a form of heuristic. Furthermore when the geometric layer is added it is interesting to take it into account: for instances the distances, and so on.

In addition another feature we could exploit is the "goal" each abstract task has: from a given state we could compute the distance to the different decomposition goals and try first the most promising solution. Indeed, as mentioned before the tasks already have an implicit goal, but the goal is also expressed as a clause containing a set of predicates which partially describe the desired state that should be reached.

The first expectation we have is obviously for this solution to be faster in most cases in the context of CTAMP. Its efficiency may of course vary depending on the number of choices in the domain but it has to be significantly faster. It is also important to compare its quality to a problem where the domain description alone would find a solution in shorter or equal time to the system using the heuristic search (possibly both systems finding the same solution). Furthermore we expect a good implementation (and formalism) to offer a negligible computation overhead.

A constraint on this new search algorithm would be to still benefit of the improved backtrack mechanism we proposed. Indeed this mechanism brings a dynamic understanding of the complexity of the problem: it reasons on the failure causes and uses them to rate the different backtrack points.

5.2.2 Principle and heuristics

The algorithm is inspired by A* and works on the HTN internal search structure: the task network and the backtrack points. We rely on the task network even though two different decompositions can lead to the same symbolic state because what matters to us is the actions used in the plan and especially their cost furthermore there is no risk of infinite loop. As the task network is structured in a tree it is possible to simplify the A* algorithm by removing the part which checks that the current "point" (search state) has not already been visited, which reduces a bit its overhead.

The previous section suggested different types of heuristics:

- A symbolic heuristic based on the distance between the current symbolic state (during planning) and task goals (expressed via the actions costs), which we call hereafter the "symbolic-projection heuristic".
- A geometric projection based and geometric computation, which could be euclidean distances and so on.
- An error diagnosis (similar to or based on the backtrack mechanism we proposed).

Another part of the heuristic is the hierarchy from the domain encoding, it is still taken into account as the list of next possible tasks during the search.

5.2.2.1 Compute the symbolic-projection heuristic and potential problems with goals associated to abstract tasks

We demonstrated in Section 2.2.1 that, at any planning time, the current symbolic state is always known (thanks to the total-order decomposition). Hence to compute the symbolic-projection heuristic toward a given task goal can be seen as a classical-planning problem: the initial state is the current state, and the goal is indeed a set of predicates. We present the way to use this heuristic, which goal we are using, in Section 5.2.3. There are several solutions to compute this heuristic, any classical state-space planning can solve it. Indeed we can use a classical planner to find a solution between the current symbolic state and the task goal and use this solution as a heuristic.

There are many classical planner already available, two illustrative examples are:

- Classical planner, such as fast downward that works with state variable as well.
- Graphplan-like or FF-like accessibility computation.

Since both are really easy to implement thanks to the extensive literature and the numerous implementations already available the best way to decide which one to use would be to do a small benchmark.

The principle of this heuristic would be to sort the different possible decompositions and variable bindings. We consider that different values for a variable can be seen as different decompositions. Hence when a method has several possible decompositions and some of those decompositions need variable bindings, the heuristic will sort all the grounded decomposition hence taking into account the value of the variables.

To exploit the knowledge encoded in the domain representation and also make this search a bit faster, the list of possible actions[3] is limited to the ones that appear in the task corresponding subtrees. Moreover this list of actions can be compiled from the domain (preventing any computation at planning time) and since it is a subset of all the actions it should reduce the complexity of the problem.

This mechanism requires that all the actions provide the list of all their effects and preconditions in order to allow the classical planner to find the solution. Sometimes domain experts write the preconditions in the methods, instead of in the action, in order to be more concise. However in our case since they are needed by the classical planner, to compute the plan that serves as heuristic, it is necessary to also have them in the actions (even if it means to have them duplicated). The purely symbolic actions must already list all their effects for the domain to be sound, however the actions with a geometric counterpart do not specify their geometric effects since they are computed by the geometric layer. So actions that have a geometric counterpart now must provide the expected effects[4].

[3] In order to allow a better control of this heuristic, the mapping we use consists in using the actions as they appear in the HATP domain. This permit the domain expert to analyse the answer given by the heuristic search.

[4] By providing the expected effect, the system would then have a way to ensure that the geometric counterpart respect a "contract": after the action has been tried in the geometry, when GTP returns the effects if an expected effect is not met then the projection is considered not valid nor is the domain encoding.

Thanks to the effects, the heuristic can even tell if a decomposition is not feasible: if it is impossible to find a set of actions that solve the relaxed problem then the actual decomposition is not feasible.

A problem that arises is the need to qualify the side effect from the actions with a geometric counterpart: in most of the cases the direct effects are obvious (e.g. a *Place* action would cause the target object to end up "on" the target surface). However the side effect such as the human affordances are not trivial to represent. In the HRI field those affordances are usually part of the tasks goal thus they can not be ignored. A solution would be to have a set of functions at geometric level capable of producing a rough estimation for a small computation cost. For instance visibility could be a computation based on the position relative to a simplified cone of view, and the reachability could be a measure of distance between the object and the agents.

When the external planner finds the solution, the cost estimation of the plan is based on cost functions for all actions in the solution. When possible the usual action cost function is used, otherwise (if the cost function depends too much on the actual state) a heuristic cost function must be provided. To have the closest estimation possible, when the action has a geometric counterpart we use the geometric projection instead. It is important to note that this heuristic returns the estimation of the cost (and not the number of actions like most similar heuristic), this allows a better understanding of what HATP is doing because it is easier to understand the impact of the heuristic. Additionally it is more straight forward to compute particularly in the context of CTAMP.

Finally an additional information that the symbolic-projection heuristic could benefit from is the set of social rules we already have: they could be applied to the solutions found by the heuristic and refine the cost of those solutions. The solutions containing forbidden states, for instance, could be seen as less interesting. All the social rules may not be relevant since the solution proposed only corresponds to a relaxed version of the actual problem (task decomposition) so the proposed allocation of the agents to the sub-tasks may not reflect the actual allocation that would be found by decomposing the task.

5.2.2.2 Compute the geometric projection

The symbolic-projection heuristic, presented above, computes a solution to the relaxed problem of reaching, from the current state, the next task goal and we mentioned that it returns the value as the sum of the costs of the actions in the solution. The geometric projection is responsible of computing the potential cost of the actions with a geometric counterpart, that appear in this solution.

For each such action, the "expected" initial symbolic state and the state modified by the action's effects will be constructed (they will only be the states according to the heuristic solution, not the actual symbolic states from the HTN decomposition obviously) from which the corresponding geometric states can be deduced. Depending on the action type and the geometric states, the geometric layer will compute an estimation of the action cost. This cost estimation must be admissible: it could be for instance the euclidean distance for actions such as navigation, or when moving object (distance between its initial and final pose).

5.2.2.3 Handle the error diagnosis

When a backtrack is required (can not project a geometric action, a plan is found and we expect to find a better one and so on) we use our backtrack-point rating mechanism. The idea is to update the heuristic found for all the current unexplored alternatives using some knowledge from the geometric "situation": we can benefit from the criteria described earlier and update the values of the alternatives accordingly. For instance the *Pick* action of a failed *Place*, and all the others. It takes into account the backtrack points that have more chances of making the solution work, hence if well balanced with the other heuristics the algorithm will decompose the most promising solution both according to its chances of geometric-projection success and for its expected cost. However this heuristic is not based on an estimation of the plan cost, it is closer to a penalty mechanism that would sort the backtrack points according to diverse criteria.

5.2.3 Algorithm

Inspired by A* we will take into account the heuristic computation for each choice HTN encounters, therefore when the decomposition process does not need any decision it proceeds with the general algorithm. The event that would trigger a heuristic computation are: the choice over the decomposition and possible variable binding, when a plan is discarded (too expensive, not feasible, and so on), or when a plan has just been found and we look for another one (a search for the best solution or a search with timeout). As soon as such a choice occurs the different parts of the heuristic are updated and combined: symbolic projection, the geometric projection and the error handling. The symbolic projection is called with the current symbolic state and the goals of all the applicable abstract tasks to determine the most promising ones: sorting their grounded decompositions (implicitly also sorting the variable binding). For each individual task, it computes the cost from the current state to its goals and use it to sort the different applicable abstract tasks. But it also filters out the task that can not be achieved: their goals can not be reached (again it requires a sound domain to work). When only one abstract task is available it will only sort the different possible bindings. The geometric projection is used within the symbolic projection to estimate the cost of the geometric actions in the solution found by the classical planner (used as the heuristic value). And finally the error handling is updated when an error has occurred, so only on some events such as and manly a discarded plan.

In order to combine the three heuristics into one, we propose to use a weighted sum. The most trivial way to decide on the weights is to start with hand-tuning. But the best solution is to learn them. However to offer the best combination possible it is important to have different weights depending on the problem class.

We could make a correspondence between a domain and a problem class, learning the weights for each domain. However we hypothesise that we could determine the classes of problem using a clustering mechanism. But the parameters of the classes must be determined, some possibilities:

- level of cluttering of the geometrical space

- geometric-planning time
- type of solution-plan found
- and so on

The “type of solution” parameter could be based on the actions and tasks contained in the solution, which reminds of the landmarks as described in Elkawkagy and Bercher (2012): “landmarks are subgoals that occur on every search path leading from the initial plan to a solution”. Then the solution found could be listed according to their landmarks and the weights could be learned for each class of problem accordingly. Later on when a new solution is requested for a given domain, the landmarks could be computed for this domain and used to retrieve the correct weights.

To further benefit from the solutions found by the heuristic mechanism, we could implement way to store those solutions. When we face a choice, we compute the heuristic and store all the solutions, then any choice that needs to be done in a task (hierarchically) under the first choice could use the solutions to bootstrap their own search hence reducing the heuristic search time.

5.2.4 Evaluation

To properly evaluate this new algorithm we must provide a variety of domains and problem instances.

- Symbolic domains where the hierarchy already gives the best solution as first solution, and check that the heuristic gives the same solution and that its computation overhead is not too important.
- Symbolic domains with several problem instances to control that the weights are retrieved correctly depending on the problem class. The classes can depend on the domain but also on the initial state.
- Geometric domains where each action projection is costly and where the heuristic could prove to be very efficient.
- If possible some problems such that the heuristic is sure to propose a first solution that is not the best, but after some backtracks still finds the best solution. Depending on the heuristic properties it can be impossible to find such problems. However it may be possible to find problems where the computation overhead is significant.

CHAPTER 6

HATP IN OTHER DOMAINS

We argue that HATP is a planning laboratory mainly designed for robotics application. In this chapter we present domains where HATP was used and allowed the cooperation of people of different communities. The first domain fall within the scope of the ARCAS project where a team of Unmanned Aerial Vehicle (UAV) equipped with a manipulator arm have to assemble structures. HATP is the symbolic planner responsible of the allocation of the tasks in the group of UAVs. The second domain, fundamentally different, consists in using the knowledge from HATP and its plans in order to have meaningful dialogue, such as plan negociation, in an HRI context.

6.1 ARCAS project

The ARCAS project[1] is an European funded project that involves multiple partners and it proposes the development and experimental validation of the first cooperative free-flying robot system for assembly and structure construction.

6.1.1 Presentation and challenges

The ARCAS project has three main axis: structure assembly by a team of UAVs, industrial inspection and space manipulation. The latter is a side-part of the project and primarily involved the control community. The industrial inspection consist in using UAVs either to visually inspect industrial plants or to carry a smaller wheeled robot that sticks to pipes for finer inspections. In both cases the problem is addressed using motion planner alone since the task is decided by a human operator (i.e. waypoints for the UAVs and so on). Finally the structure assembly is the part combining work on assembly sequence planning, task planning, motion planning and execution architecture (control, collision

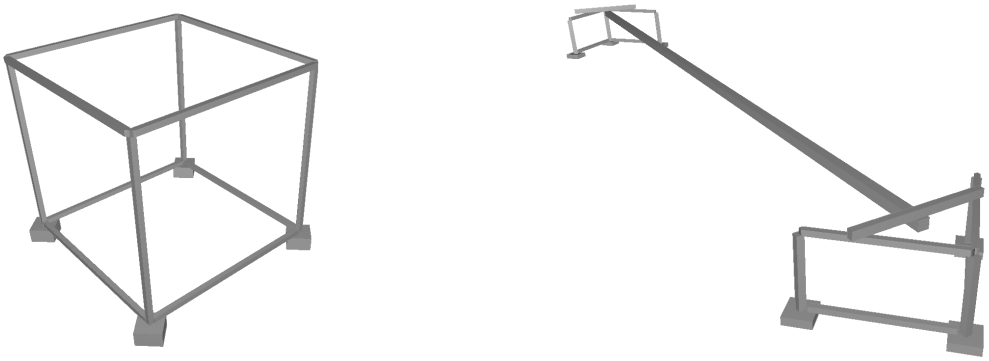
[1] Webpage of the project: http://www.arcas-project.eu/

avoidance, and so on). The system developed can work with different UAVs, the three platform used are presented in Figure 6.1.

(a) Indoor UAV for light loads (b) Outdoor UAV for medium loads (c) Outdoor UAV for heavy loads

Figure 6.1 – The three platforms used in the ARCAS project. Both UAVs from 6.1(a) and 6.1(b) are quadrotors with eight propellers while the one from 6.1(c) is a Flettner helicopter (bi-rotor helicopter).

To have simple yet interesting structures to assemble the project focused on structures made of bars. Some examples can be seen in Figure 6.2. The simplicity comes from the clipping mechanism: when two pieces are brought together they clip ensuring a strong link. On the other hand the complexity comes from the need for cooperative transport of certain long bars requiring two (or more) UAVs to strongly cooperate. Moreover the robots are using arms which must be compliant to avoid any problems. To carry out the assembly of the structures, the complete system must exhibit a set of properties: multi-robot plans, cooperative transport, perception and localisation and visual servoing.



(a) Cube structure (12 parts) (b) Long bridge structure (15 parts)

Figure 6.2 – Examples of structures from the ARCAS project. In 6.2(a) there are 12 parts composing a cube, but there also exist a variant with four additional diagonal bars for a total of 16 parts. In 6.2(b) the last part to assemble is the long bar linking the two supports together.

In the project, the planning level was composed of three planners: (1) an assembly sequence planner that computes the assembly sequence from the assembled structure, (2) the symbolic planner, HATP, is responsible for the task allocation, and finally (3) a motion planner, also developed at LAAS, to compute the trajectories for each action in the symbolic plan. At the beginning of the project we linked the three planners in a linear workflow, the output of a planner was the input of the next, in a three-layer fashion. The output of the complete planning level is then used by an execution architecture that supervises the execution of the tasks and ensure that there won't be collisions or any other problems. Later when the integration between HATP and GTP was ready we used this combined planner, see Figure 6.3: it allows to test the feasibility which is very important in the context of ARCAS because the environment changes significantly as the structure is built. Moreover the robots handle bars that can be long enough to risk collisions with the structure.

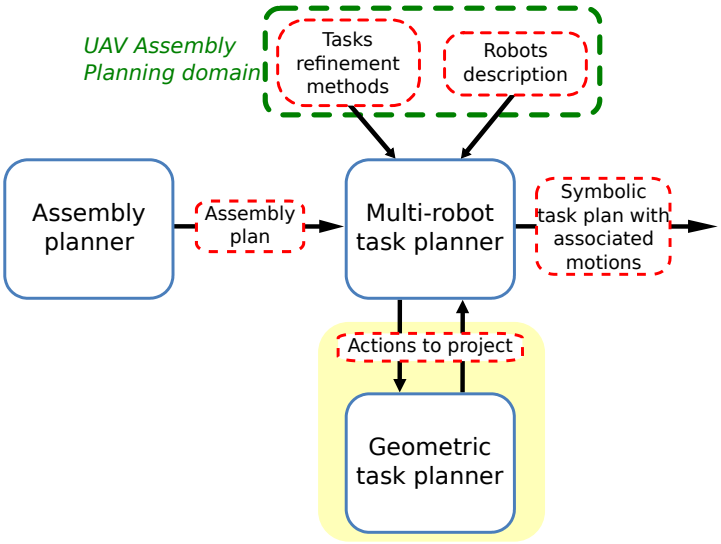


Figure 6.3 – Integration of the planning level in ARCAS using SGP. The assembly planner gives an assembly plan, a dependency tree listing the operation to assemble the structure. The Multi-robot task planner, actually HATP, takes the domain description and the assembly plan to produce the symbolic plan and allocate the actions to the agents. GTP is called to assess the feasibility of the actions and the trajectories found are kept along the actions of the symbolic plan.

Integrating the assembly planner and SGP together, we were able to compute symbolic plans with associated trajectories that could be executed on the system. However there was a combinatorial problem, since the assembly plan was given as a dependency tree where each part would specify what other part must be assembled before the symbolic planner had to find an actual assembly sequence. For instance in the cube structure, Figure 6.2(a), the four first bars that compose the first level can be assembled in $4! = 24$ different orders. With the two other levels this number explodes, indeed the worst case scenario is $n!$ where n is the number of parts in the structure. The solution we propose is to have a tighter integration between SGP and the assembly planner.

6.1.2 Symbolic-Geometric planning in UAV context

Figure 6.4 shows how our complete planning system can assemble structures. In this version the assembly planner gives the assembly plan, then HATP uses it to choose which structure to assemble first. Without geometric projection there would be no difference at symbolic level while in fact if the big (blue) structure is assembled first then the small (green) one can not be assembled because the robot can not reach the position to place the last horizontal bar. So HATP tries both order and realises that the order with the small structure first is the only one that leads to a feasible plan.

6.1.3 Integration of SGP with assembly planning

To tackle the combinatorial explosion of the assembly sequence, instead of relying on HATP that is not exactly suited for this task, we integrated the assembly planner with the already combined symbolic and geometric planners. Indeed the assembly planner has all the tools necessary to reason about the assembly sequence as we illustrate in this section, this new integration is exhibited in Figure 6.5 and was first introduced in ARCAS Deliverable D6.4 (2015).

6.1.3.1 Principles

To present the new architecture, we must introduce more details about the assembly planner, developed by the University of Seville. The planner takes as input a model of the structure as it should be assembled, and computes the assembly sequence using an assembly-by-disassembly approach: it tries all the ways to disassemble the structure to guess the assembly sequence and the dependencies between the parts. To improve this search two additional tools are used, a physics engine to ensure the stability and self-sustainability of the structure at each step, and a heuristic search.

The heuristic used consist in computing the risk of collision in order to find an assembly sequence with less chances of collisions (self-collisions between the parts or collisions with the obstacles). To obtain this information, bounding volumes are placed around the objects and collisions between those volumes are used. Once a solution is found the assembly sequence is sent to HATP along with the dependencies between the parts. Thanks to the heuristic search this assembly plan is supposedly the most likely to succeed.

It is possible when solving the problems individually that solutions to one problem are incompatible with the other domains (e.g. it might not be possible to compute paths for an otherwise correct assembly plan). Thus, it is important to keep a loop of information between the planners. This loop can also hint each planner into the right direction to get better plans. For example, information from the symbolic planner can help the assembly planner to choose a sequence of operations with better odds of succeeding on later phases of planning. At the other end of the system, a complete assembly plan can save the symbolic and geometric planners a good deal of computation, because it provides them with a smaller search space, with far less degrees of freedom.

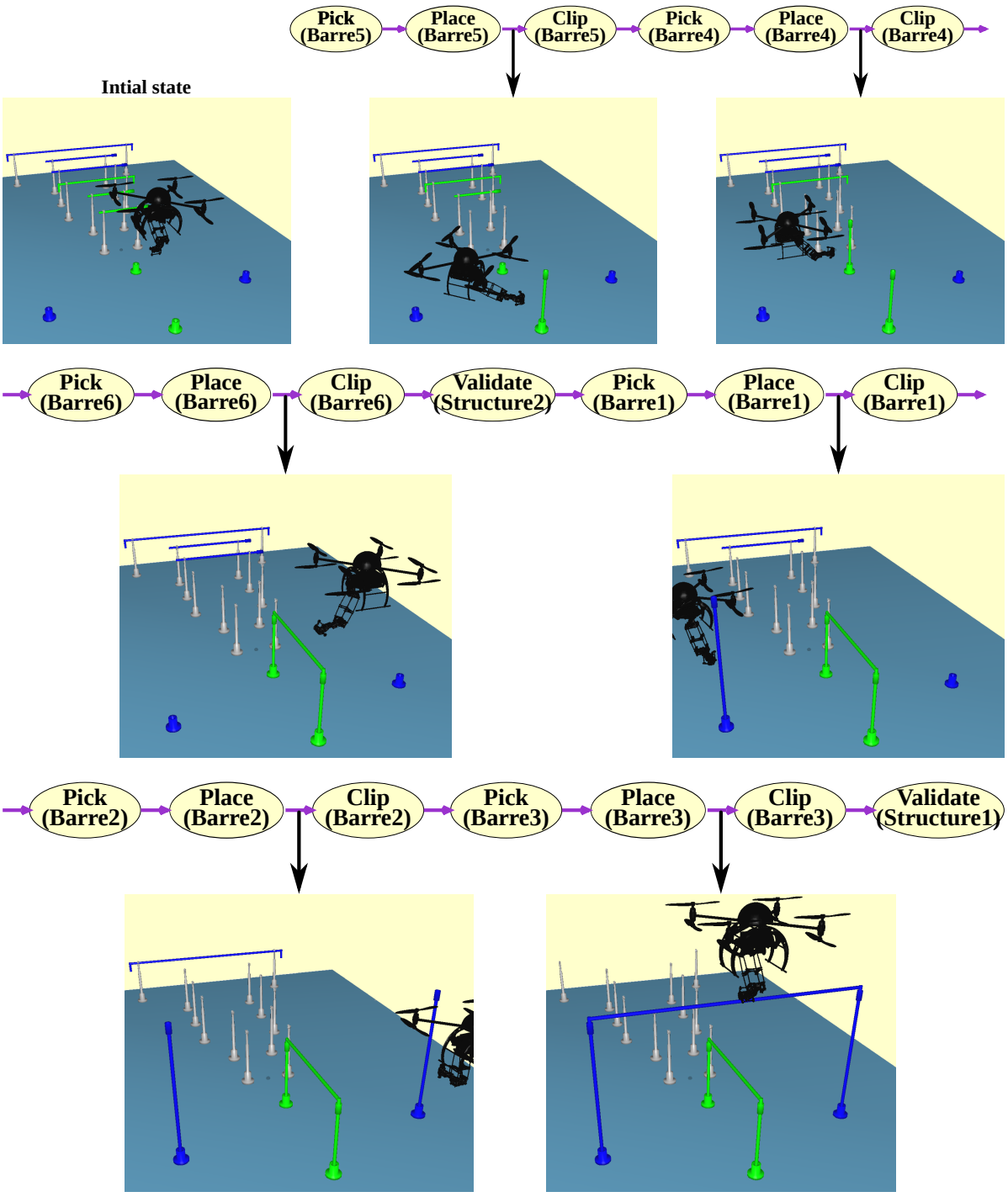


Figure 6.4 – Complete assembly of two bar structures with a UAV, if the planner decided to assemble the blue first it would be impossible for the green to be finished because the robot can not assemble the last one without being in collision with the long blue horizontal bar. *(The black arrows are pointing at the geometric state corresponding to the symbolic state.)*

The approach followed in this work, illustrated in Figure 6.5 is to feed a structure to the assembly planner, which will generate a single assembly plan, and to give this plan to the symbolic planner, which in turn will invoke the geometric planner when needed. The assembly planner is guided by a heuristic that can be modified with information from HATP. When an assembly plan is detected as infeasible during HATP or GTP phases, information about the causes of the failure can be fed back to the assembly planner to improve the heuristic and have better chances of finding a successful assembly plan.

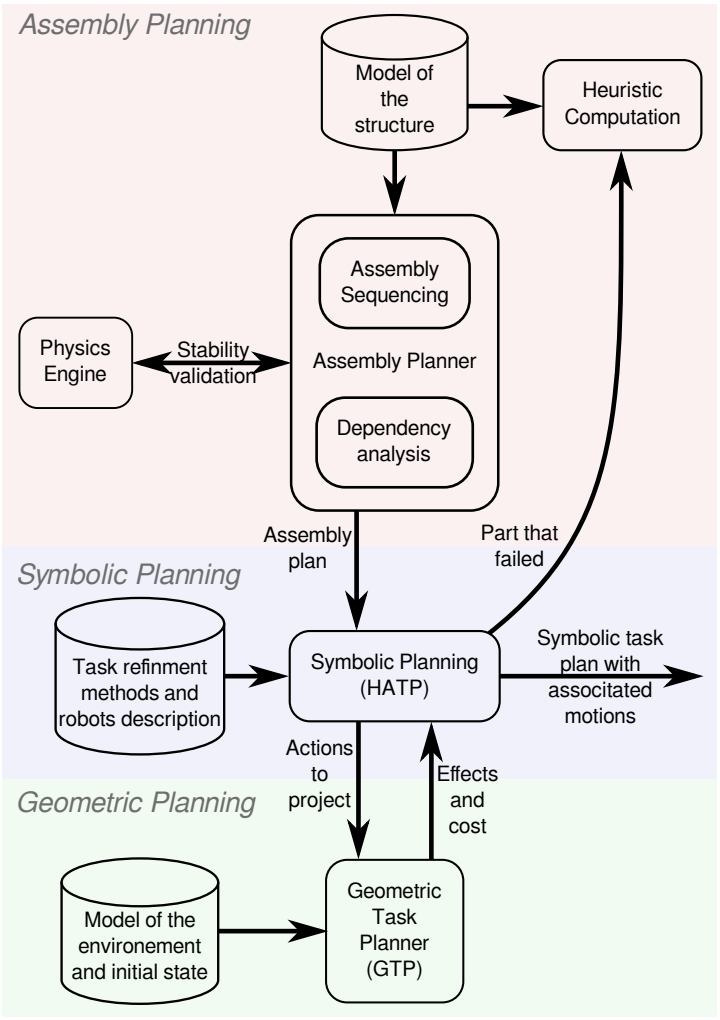


Figure 6.5 – Complete planning level combining the assembly planner with SGP. The assembly planner sends the assembly plan computed thanks to the heuristic, then HATP plans the symbolic action and assesses their feasibility at geometric level thanks to GTP. If the symbolic plan can not be found or if it is not geometrically feasible, the part that prevents it is sent to the heuristic that is updated and a new assembly plan is computed.

We decided to send only one plan at a time, in order to reduce the search space HATP has to work with. Sending an explicit graph representation (e.g an AND/OR graph) of all plans is impractical for

big structures in general. The number of possible states follows a growth rate of the order of $O(n!)$ with n being the number of pieces that comprise the structure. Hence, a simple structure as the cube used in this work (16 pieces) can easily get a graph with too much nodes to be able to compute it, reaching the trillions. In addition to that, evaluating each plan has an important cost, because we have to try it in both GTP and HATP, which can take some time.

Our approach not only reduces the search space for both planners, it also allows each of them to address just the part of the problem it fits better. For example, GTP would have a hard time figuring out stability issues within a structure, but the assembly planner is specifically designed for that. Complementarily, the assembly planner can only solve linear straight paths for assembling each piece, but GTP can find more complex solutions if they exist. While geometric analysis might be used to predict some precedence relationships between assembly operations that must be fulfilled, it misses a family of limitations of mechanical nature such as balance and friction issues (where extracting a piece in a given direction might drag other pieces out of place). The case shown in Figure 6.6 demonstrates a structure where pure geometrical reasoning can not anticipate the cause of instability, since the behaviour depends on physical properties of the bodies (i.e. their masses) and there is no direct contact between the critical pieces. Since the assembly planner performs rigid body simulations on the resulting subassemblies, the problem would be detected early, and would never reach the HATP or GTP phases, thus saving a lot of computation time that would be wasted otherwise. Note that this is different from physics based geometric planning, since we are studying a stability condition inherent to the structure itself and independent from the path that agents follow while actually executing the assembly sequence. In a similar fashion, HATP can handle multi-agent domains and allocate tasks in parallel easily, while adding this to the assembly planner would increase complexity and computation time.

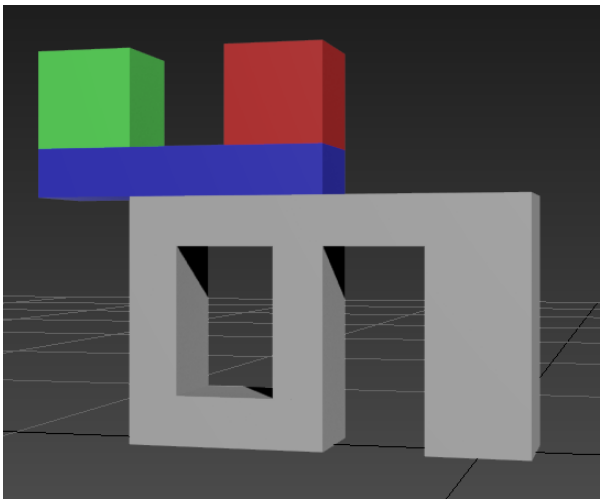


Figure 6.6 – Example of structure with potential instability. The stability depends on the order of operations where there is no contact between critical pieces. If the red block was to be removed before the green one, the structure would become unstable and the blue block would fall.

6.1.3.2 Formalisation

A planning session starts with HATP requesting an assembly plan from the assembly planner for a given structure. Such request would simply contain the name of the structure that it needs to analyse, and a request id that must be the same for further requests in the same session.

The assembly planner will retrieve the current world state (mainly obstacles) from the same model that GTP uses, and a description of the structure as it should be (from a 3D model) and generate an assembly plan for it. Then it will answer with a response in the form of a tuple $PR = \langle O, D \rangle$, where:

- $O = \{o_1, \ldots, o_n\}$ is the ordered sequence of assembly operations o_i that must be performed to assemble the structure. Each element o_i describes the type of operation and the pieces involved. At the moment, the only supported type of operation is to place the piece in its support, but this can be generalized to other operations like gluing, screwing them in or securing them in other ways. An operation has the form: $o_i = (id, type, \{pieces \ldots\})$, where the first element is a number to identify the moment when the operations should be done (the smaller the number the earlier it must be done), then comes the type of the operation, and finally the list of pieces.
- $D = \{d_1, \ldots, d_m\}$ is a set of dependencies between assembly operations. Each element d_j is a tuple made of the target operation and the set of all other operations that must necessarily be performed before that one, with $d_j = \langle t_l, p_k \rangle$ which reads as the set of operations p_k that must be achieved before the operation t_l, where t_l is just the identifier of an operation o_l. The set p_k contains the list of identifier for operations from O.

Making explicit precedence relationships between operations allows the symbolic planner to perform more than one operation of the sequence at the same time in multi-agent environments, with one or more assembly operations starting before all previous operations in the sequence are finished.

See below an example of such plan, for the cube in Figure 6.2(a), where in order to assemble the vertical bar we need the four operations that put the horizontal (bottom) bars:

$$PR = \langle \{(0, place, horizontal\_bottom\_bar\_0), \ldots (4, place, vertical\_bar\_5), \ldots\},$$
$$\{\langle 4, \{0,1,2,3\}\rangle, \ldots\}\rangle$$

When HATP starts, it requests the assembly plan and upon receiving it starts the decomposition. The first step is to detect which operation to achieve first and to do so it uses the identifier, starting at 1 and going up to the number n of operations. For each operation an *Assemble* task is called: it detects the type of operation and execute it for all the parts listed. Then it uses the dependencies D to add causal links and in multi-agent scenario uses those links to parallelise some operations.

6.1.3.3 Heuristics

The assembly planning search space is tremendously high even for structures with a small number of pieces (ten to twenty), so we use a heuristics based best first to navigate it efficiently. While choosing

heuristics to guide the assembly planning process, the goal is to reach an assembly plan with high probability of being successful during the symbolic and geometric planning phases. By feeding information from the later phases back into the assembly planner's heuristic, the odds are improved but both domains remain decoupled.

Different heuristics can be used in assembly planning for this purpose. We used a measurement of the free space available around each piece in the directions of assembly, in order to give preference to the least occluded one. The reasoning behind this heuristic is that occluded pieces are more likely to introduce difficulties during assembly by posing obstacles to the agents. For any given piece, its geometry is projected along various directions (possible assembly directions) and each collision detected will increase the cost associated to the removal of that piece. If replanning is needed, these costs can be increased according to information received from the symbolic planner. In this fashion, occlusion would be more costly if the associated piece is directly linked with the reason why the first plan was discarded.

For future work, other heuristics such as using a bounding sphere of the size of the agents may be tested.

6.1.4 Multi-UAV planning

The current implementation of GTP has a limitation: it does not allow to plan trajectories in parallel, it is only possible to plan trajectories where no other robot move or cooperative transport (which is considered by GTP as one "agent" acting, not two individual trajectories). To cope with this limitation we generate fully sequential symbolic plans that can later be transformed into plans with parallelism thanks to a post-process, as long as the dependencies between the parts are respected.

The actions that GTP can plan are: **goto**, **pick**, **place** and **pickAndPlace**. The latter is more efficient than a pick followed by a place because the different step to compute the trajectories, grasp and position are interleaved. (It is still important to allow separate pick and place if we need to add actions in between.) In addition to those actions with a geometric counterpart we have a *monitor* action, which triggers in the supervisor a visual servoing low-level task. It could be interesting to plan it in GTP, so the position of the monitoring UAV is taken into account when planning the others trajectories, but again GTP can not plan actions in parallel. Consequently there is no point in planning the monitoring trajectory, so in our current implementation it is just considered a symbolic task during the planning process.

Figure 6.7 presents an extract from a plan where a monitoring task must be carried out in parallel of an assembly task. When the robot must place the part, the action is actually called *PlaceInSupport* because of the domain representation we use when HATP and GTP communicate to assemble parts. The idea is that HATP needs a way to tell GTP where to place the bars, and we refer to the end position as "supports". Those supports are extracted from the final configuration of each part in the assembled structured (from the same model the assembly planner uses). In this particular example the interesting point is the causal links, they force the monitoring task to last as long as the assembly

task (place in support).

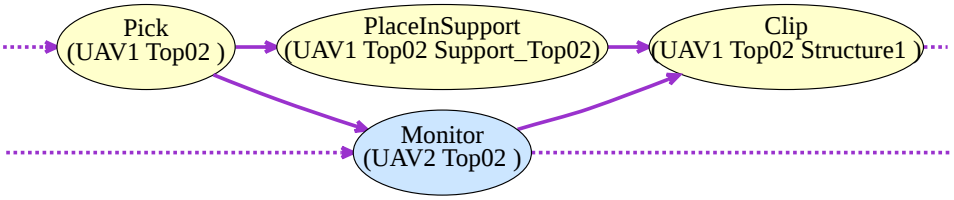


Figure 6.7 – Extract of a plan with a monitoring task. The robot UAV1 picks and places the bar Top02, while the UAV2 monitors this assembly.

Figure 6.8 shows another extract from a plan, this time however it is focused on the action necessary to execute a cooperative transport. Our strategy is to elect a master and create a group, each other UAV necessary is added to the group. To assign all the UAVs in the group (master and others alike) we use the payload of each UAV and compare the sum to the masses of the parts to transport. However if we only limit ourselves to this comparison, we will end up trying many group just by changing the order of the UAVs in the group (for instance a group with UAV1 and UAV2, and another UAV2 and UAV1, two different group for HATP, but with the similar meaning). To test all the possible orders is useless since the order of the UAVs in the group does not change the efficiency of the group. Moreover it represents a very big number of attempts: $u!$ in fact, where u is the number of UAVs in the problem. To address this issue we have a mechanism that stores the group that were tested on each part, and prevents to try the same group with a different order on the same piece.

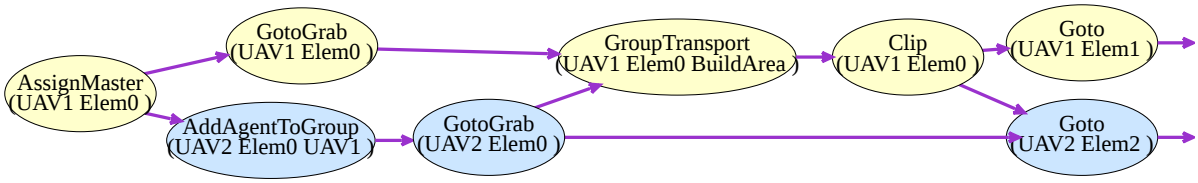


Figure 6.8 – Extract of a plan with a cooperative transport. In order for a group of UAVs to cooperatively transport a bar we must assign them into the group, then they must all go to the part and grab it, then the group transports the bar. When it is finally clipped to the structure only then the UAVs are released and can go do other actions.

To summarise, thanks to GTP the geometric feasibility of the actions is ensured, and thanks to the physics engine from the assembly planner the stability of the structure is guaranteed at each step of the assembly. The symbolic planner adds monitoring actions to increase the chances of success when actually executing the plan. The symbolic solution plan has a motion associated to each action, but is completely sequential. However it is possible to have parallelism in the plan thanks to a post process as long as the dependencies between the parts are respected.

This work was realised with Carmelo Jesus Fernández Agüera Tortosa and Alexandre Boeuf in the scope of the ARCAS project. The former, from University of Seville, took care of implementing

the assembly planner. The latter, from LAAS, worked on specialising GTP to work with UAVs and allow structure assembly in the motion planner. We handled the design of the new interface with the assembly planner and the description of the different domains to work with several UAVs. A variety of domains were necessary to test the different parts of the system but also because GTP can not work with multiple UAVs flying in parallel. So we had to design domains with the assembly and symbolic planners to work on the multi-UAVs aspect, and some other domains where GTP was used to assess the feasibility of the assembly plan.

This concludes the presentation of the work achieved in the scope of the ARCAS project. The next section focuses on the usage of HATP in the context of dialogue interaction with a human user, the objective is to increase the mutual understanding and allow a finer control over the repartition of the task both according to the user's preferences and adapted to its level of expertise.

6.2 HATP in dialogue scenarios

One of the great challenges in robotics is to build a system able to collaborate with humans. Collaborating with human partners requires for the robot to plan its actions and anticipate its human partner contribution to the overall plan, to monitor humans' inherently complex actions and to maintain a model of the world state. These interactions should also be performed in a socially acceptable manner in order to ensure human's comfort as well as legibility and acceptability of the robot behaviour.

We want to share a plan with humans in a context where the robot does not know exactly the knowledge and desire of its human partners. So we want to offer a way for the robot to verbalise and explain the plan so the humans can understand what they are expected to do. However the humans may object with some tasks they have been assigned, so we offer a way to negotiate the plan so the humans can specify their preferences.

6.2.1 Introduction

We define a collaborative plan, or shared plan, as a set of linked actions involving several agents that cooperate towards the same goal. To generate a shared plan, the robot should take into account not only the environment configuration but also its human partner. A common way to do so consists in computing the affordances of the partner. On a social level, the robot should also be able to adapt the plan to the user's preference and knowledge on each task of the plan.

Once the robot has generated a plan to achieve the common goal, it needs to be shared with the human partner in order to ensure that they are aware of the tasks they have to carry out, and that they agree to perform them. When dealing with simple plans, infants can cooperate without using language if the plan and goal are simple enough. In situations requiring more complex ones, language is the preferred method Warneken et al. (2006), Warneken and Tomasello (2007). However explaining the whole plan at once would be inefficient and annoying for the human partner, especially if she/he is already aware of the way to achieve parts of it.

Research in psychology and philosophy have led to a better understanding of human behaviours during joint action and collaboration, to know how Tomasello et al. (2005) and why Tomasello (2011) we collaborate and what is shared Butterfill and Sebanz (2011).

This research on joint actions was used in robotics to perform tasks involving human partners. While a substantial number of papers address the issue of plan generation for collaborative goals involving a human partner, only a few actually study how to efficiently adapt the plan generation and its execution to the user's knowledge. We also believe that representing and maintaining such information correctly during the interaction is a key issue. We argue that this user adaptation would significantly improve the social skills of the robot during the interaction.

We will first explain how we track human's knowledge on each HTN node, from high abstraction-level tasks to the level of actions. Then we present how we are able to take the human into account during collaborative plan generation. We then provide details on how our method uses the tree structure of the generated HTN to: (1) present and negotiate the shared plan, and (2) explain and monitor tasks according to the user's knowledge level, in order to guide or teach the human when needed and to adapt the monitoring of their actions.

6.2.2 Human-knowledge tracking

6.2.2.1 Situation Assessment and Mental States

To assess the knowledge state of its collaborator, the robot needs to understand the situation and extract information about agents. To do so, and to maintain a consistent world state, we use a situation assessment component to perform spatio-temporal reasoning based on data about humans, robots and objects, in similar manner as Milliez et al. (2014). It also computes affordances for each agent (reachability and visibility). This world state will be used by our plan generator to compute a plan adapted to the situation.

Using situation assessment, in our previous work we successfully manage a belief state for each agent, robot and human. Each belief-state model is independent and logically consistent. The robot belief regarding its counterparts' mental state about the environment is represented in these models.

This section is focused on human's knowledge on tasks. This knowledge is represented as the vector <HUMAN, TASK, PARAMETERS, VALUE>. HUMAN is the human having this knowledge, TASK is the name of the task, PARAMETERS is the list of relevant parameters to describe the task knowledge (see below) and VALUE is the value (or level) of knowledge.

As an example, the fact that the *human1* has an *expert* knowledge on assembling a furniture piece A with a piece B would be represented as: <*human1, assemble, [A,B], EXPERT*>. The possible knowledge values are *NEW*, *BEGINNER*, *INTERMEDIATE* and *EXPERT*.

Some tasks can be considered as common knowledge. For instance, putting ingredients in a bowl is considered simple enough to be a known action for any human. This kind of task will then be tagged as common knowledge and considered as known by the users no matter the parameters. Some other tasks knowledge might differ according to the parameters. For these tasks, the knowledge may

be linked to a "type" of parameters instead of an instance of this class. As an example, we can consider that if a human knows how to paint the living-room, she/he will know how to paint any room. In this case we will put in their knowledge the type "room" for the task of painting instead of the instance living-room. To sum up, some tasks can be tagged as common knowledge while other tasks can be described by some of their parameters or parameters type. This formalism of task representation requires a domain expert to indicate how to represent the tasks knowledge.

6.2.2.2 Knowledge levels on task

In this context, as the robot generates the shared plan, we assume that it knows all the tasks in the plan. Concerning the collaborator, we define four task-knowledge levels that will lead to different behaviours from the robot.

- *NEW*: this first value will be used for tasks which have never been performed by the user. If the user observes the task being executed with explanation or if he performs it himself, the value will be changed to *BEGINNER*. However if the user observes the task being executed without any explanation, we keep the level as *NEW* since we consider that she/he has not been given enough information to link the observation with the task.

- *BEGINNER*: this value will be used for users who have already achieved the task but may still need explanation to perform it again. If the user successfully performs the task again, without asking for explanation, the value is changed to *INTERMEDIATE* and to *NEW* otherwise.

- *INTERMEDIATE*: this value will be used for users who are able to perform the task without guidance. If the user successfully performs the task without guidance again, the value is changed to *EXPERT*. In case of failure, it is downgraded to *BEGINNER*.

- *EXPERT*: this knowledge level will be used for users who are able to perform the task without guidance and are experienced enough to explain it to a third party. If the user fails in performing the task, she/he is downgraded to *INTERMEDIATE*.

These tasks knowledge levels allow to adapt the collaborative plan generation, explanation and monitoring, as shown in the following sections.

6.2.3 Human-adaptive HTN planner

To use HATP in the context of dialogue proves to be interesting since it offers a better understanding of the context in which an agent is asked to carry out an action. This context is beneficial for plan explanation. Indeed, it implements an iterative context-sensitive refinement process. Hence, the system can guide users by telling why they should perform an action and how it is linked to previous and next parts of the plan. All this information can be directly extracted from the hierarchy.

Once the best plan (note that one can limit the search to a "sufficiently" good cost level) is computed, it is sent to the supervisor in the form of an HTN tree decomposition and the agents stream. In addition to the causal links that ensure the actions sequencing and synchronisation between agents, the plan may include joint actions allocated simultaneously to two or more agents because they need tight collaboration (e.g. handover action). Figure 6.9 depicts a part of the tree decomposition of a solution plan.

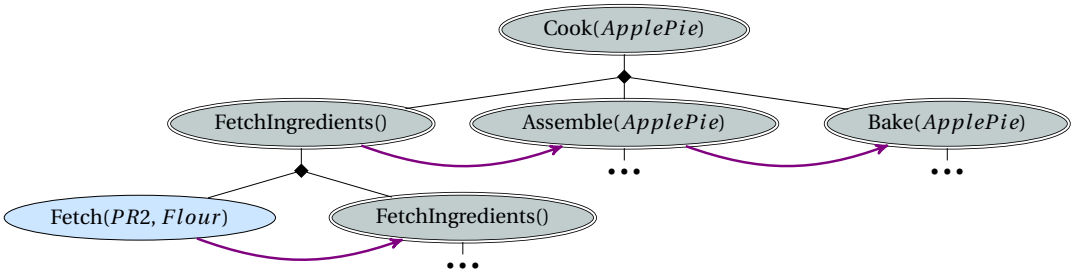


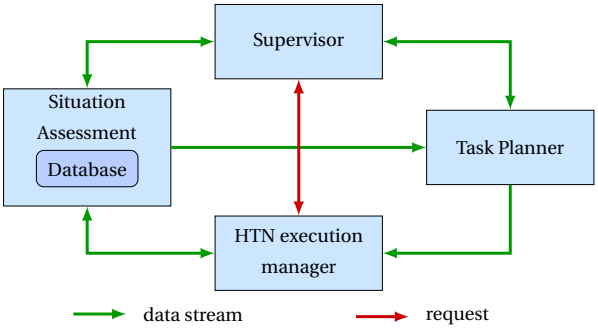
Figure 6.9 – Extract of a decomposition tree to cook an apple pie.

To take knowledge into account while planning, a new social rule was needed. The aim is to select the best suited plan for the given policy. We propose two policies: favour teaching, so the human can learn from the robot, or efficiency. With the teaching policy, the planner tries to produce plans maximizing the number of human tasks where they have the opportunity to learn, while efficiency makes the planner select plans with the least amount of unknown tasks for the human in order to ensure that they can be more efficient. In case of a policy to favour efficiency, the rule is simply to apply a penalty every time the human has to perform an action they ignore. This penalty would be reversed for the teaching rule.

The implementation is straight forward: once the plan has been computed, we apply a penalty on each task according to the policy. For instance if the human knows how to bake the pie but not how to prepare it (*Assemble* it) and we want to be efficient, then we will put a penalty on each action on the plan where the human must prepare the pie. Hence by calculating all the plans and applying this social rule we can sort the plans by their value and we will obtain the plan with the least learning possible. However should the robot be unable (e.g. inappropriate tool) to prepare the pie then the human will have to do it and the robot will have to teach and supervise this task. (We present later a way for the human to specify her/his wish to accomplish certain task by negotiating the plan.)

To illustrate our planner and the new social rule let us consider the toy example where a human and a robot have to cook an apple pie, see Figure 6.10. A part of the solution plan is shown in Figure 6.9. In this context we can consider that the human knows how to carry out all the actions (pick, place, cut, and so on) but he/she may not know the exact order of steps (higher level task). If we favour teaching, the plan should contain a way to achieve the recipe with a minimal knowledge level on each task and, as much as possible, human will be in charge of those steps. On the other hand if we favour efficiency the plan should contain the smallest amount of unknown tasks to be performed

by the user. Using this rule, the robot is able to adapt its plan generation to the knowledge of the user concerning tasks contained in the shared plan. To properly compute the cost of a plan, the planner will also consider a task knowledge as upgraded once it is added to the plan. This allows, for efficiency policy to prefer plans that reuse a same task many times and assign it to the same user to lower the cost, over some plans where different tasks are performed or a same task is performed by a different agent.



(a) The architecture used in this dialogue domain.

(b) The scenario where a human and a robot have to cook.

Figure 6.10 – The dialogue scenario, where a robot and a human must collaborate to bake an apple pie. However the knowledge the human has on the recipe depends on the user, so the robot must adapt the verbalisation of the plan to the user level of knowledge. The system also adapts the monitoring: being more careful with a beginner, and leaving more flexibility with an expert.

The planner is integrated in a dialogue system that allows to negotiate plans (see below). More precisely it allows to ask about user preferences and abilities. If the user tells the system that she/he cannot perform a given task, it will not be added to the plan (invalidate the corresponding task precondition). Concerning user preferences, the negotiation step will update a database with the preferences expressed by the user. If the user specifies that she/he (does not) want to perform certain tasks, those tasks if added to the plan will take an important reward (resp. penalty) cost. Hence plans which contain such tasks will be considered as unwanted, however if they are the only possible solutions (because of inability and so on) they will be kept and the planner will return the one with the least number of unwanted and maximum number of wanted tasks.

6.2.4 Shared plan presentation and negotiation

Once the planner has generated a collaborative plan adapted to the chosen policies (about learning, abilities and preferences), the plan must be shared with the human partner. Speech is a “potent modality for the on-going maintenance of cooperative interaction” Lallée et al. (2013). Indeed,

Tomasello even suggests that the principal function of language is to establish and negotiate cooperative plans Tomasello et al. (2005). Considering this, we decided to use speech to present the plan to the collaborator.

6.2.4.1 Plan post-processing

The HTN tree generated represents a solution to achieve the goal. However, it may not be suitable to present or explain it like it is to the collaborator as it may contain refinement steps that would make the explanation unclear. To adapt the plan for explanation we use two rules. (1) we remove the recursive tasks. If a node n of the HTN tree contains the same method (using $compare$ function) as its parent $parent(n)$, it will be replaced in the tree by its children $children(n)$. (2) we also replace nodes with a single child by their children.

1. **if** $(compare(n, parent(n)))$ **then** $n \leftarrow children(n)$
2. **if** $(children(n).size() = 1)$ **then** $n \leftarrow children(n)$

These rules build a lighter plan tree to process.

6.2.4.2 Plan presentation

Before executing the plan, the robot will present the goal and the proposed allocation of high-level tasks to give a global view on the plan. Standard Natural Language generation is used as shown in Table 6.1. To ensure the scalability of the system, when presenting the plan, the robot will limit itself to verbalizing only the N first highest level tasks. For simplicity, we have chosen $N = 3$ based on some runs carried out during the development process. However we believe that this number would require further investigation depending on the domain or on the user and her/his confidence in the execution of the tasks. The robot will present the first steps of the plan, and then execute them. Once this execution is achieved, it will repeat the present/negotiate/execute process until the plan is completed or aborted.

agents(root) + have_to + root	"We have to cook an apple pie."
introduce_presentation	"I will tell you the steps."
agents(child[0]) + first + child[0]	"I will first fetch the ingredients,"
then + agents(child[1]) + child[1]	"Then you will assemble the apple pie,"
finally + agents(child[2]) + child[2]	"Finally, you will bake the apple pie in the oven."

Table 6.1 – Presentation of a plan to cook an apple pie. Root is the root of the tree and child is a list with its children. This test corresponds to the tree structure partially presented in Figure 6.9.

6.2.4.3 Plan negotiation

Once the robot has presented the main tasks and the tasks repartition, it has to ensure that the human agrees with this shared plan. The robot will simply ask the human for approval and inquire what is wrong in case of disagreement. In the current version of our system, two kinds of human requests are handled. First the user can express her/his preferences, either the will to perform a task previously assigned to the robot or the denial to perform a task assigned to her/him. The other possibility is to inform the robot that the user cannot perform an action. This will be added to the user's model and stored in the database. HATP will then try to find a new plan that prevents the human from performing a task the user is not willing or able to perform. This plan will then be presented and the robot will ask again for the user's approval. In our system, the user's preferences have a higher cost than the teaching policy as we consider that the user should have the final decision.

This mechanism is implemented the same way the social rule about learning/efficiency is: after the plan has been found, it applies a penalty to each task that the human must do when the human specified her/his will not do it, and it applies a penalty when the robot does a task she/he wanted to do. Thus a plan with a high cost means that some of the user's wishes are not fulfilled, however sometimes the best plan is not the one the human wanted, the robot may not be able to do the task the human did not want.

6.2.5 Adaptive plan execution

6.2.5.1 Execution and monitoring

Once the current task to perform has been explained, the robot performs it if it is allocated to it, or it monitors its human partner's task performance. Consequently, the monitoring may be done at high-level tasks if the human has enough knowledge of it. We have chosen this adaptive way of monitoring since we believe the robot would be more efficient, sparing its resources, by focusing its attention more often on parts of the plan that have never been performed by the human partner, and less when he has some form of expertise, leaving more freedom to the human on the way to execute tasks he knows.

Monitoring human actions is complex, particularly with high-level tasks, where we are not monitoring a set of atomic actions. The system should have reasoning models for the robot to understand if the state of the world is coherent with the action that the human needs to perform. It should also be able to measure the level of engagement of the human to the task, in order to better assess if the human is executing or not his part of the shared plan, and to react accordingly.

6.2.5.2 Failure and replanning

The goal of monitoring human actions is to be able to recover from unexpected behaviours from the human. When this happens, the robot needs to inform the human of his potentially wrong behaviour and downgrade his knowledge level. Consequently, the next time the human will perform this task

the robot will guide and monitor his execution at a more refined level (the children tasks). The task planner is requested to compute a new plan to achieve the goal with the updated world state. One of the benefits of the dynamical update of human knowledge is that this new plan may have tasks that the robot has already explained or that the human has performed before the failure occurred. In this case, guiding the human through the new plan execution will be faster as the robot will not have to reexplain these tasks. This replanning behaviour gives robustness to the robotic system and allows a socially acceptable recovery procedure where we inform the human with the error and reexplain the plan only at the needed level of detail.

6.2.6 Conclusion

A user study was conducted with this implementation and is presented in Milliez et al. (2015) along more details on the architecture used. The goal was to test how users evaluate our system adaptability, the study was carried out on a small group and only through an online survey with video clips but already proved that the robot was perceived as smarter. As future work we envisage to conduct this users study on a real robot as we believe it would lead to more realistic opinion.

In this section we presented a solution for a robot to manage a human-robot collaborative activity from plan generation to execution, guiding its human partner in an efficient and socially acceptable way. Our approach is based on a dynamic tracking and online updating of the partner's knowledge. The proposed approach offers two policies, teaching or efficiency, which provide different levels of interactions. Our method incorporates the monitoring of high-level human actions, focusing on the completion of tasks instead of the details on how the task is completed. This provides some flexibility to the human when performing his assigned tasks.

All the work presented in this section was done with Grégoire Milliez, while he brought the knowledge about dialogue management and situation assessment we were responsible of the task planning aspect. We offered the understanding of the planning process and of the plans structure which allowed to identify pattern to verbalise the plans. To represent the user knowledge we decided not to change HATP core but to encode it in the domain description since it was enough for our use case, but if the work was to be extended it would require HATP to encode the agents knowledge at planning level.

CONCLUSION

In this thesis we study task planning in the context of robotic problems. We have based our work on the Human-Aware Task Planner, a total order HTN planner and extended it. It is now called the Hierarchical Agent-based Task Planner. This planner extends the HTN planning process with new features adapted to the robotic field.

Even though there are more recent and more powerful symbolic planning methods, we focused our work on HTN because it offers a hierarchical representation of the problem which allows a better understanding and with good encoding we can have reasonably efficient planning time. Another reason is that our planner has already been extensively used in robotic domains and is endowed with features dedicated to solve problems that classical planners are not optimized for.

First of all it features a user-friendly language that allows people from many background to understand and manipulate domain descriptions. However the most interesting features are focused on the search algorithm, HATP is capable of planning for several agents splitting the solution into streams of actions (one for each agent), but it also uses a cost driven search allowing to prune out plans that are not as good or better than the current best solution. Since HATP is often used in Human-Robot Interactions it is embedded with tools to reason about the social norms: it can split the effort, prevent wasted time and many other rules to produce more socially acceptable plans.

However this approach does not allow to reason about the actual outcome of the actions on the world state. In some cases the problem is not constrained enough to be problematic to the supervisor, and it could solve the deviation on the fly still following the symbolic plans. But as soon as the geometry becomes more complex, and even more when considering affordances, reasoning just about the symbolic plan proves to be inadequate: many situations occur where the system can no longer execute the plan because of cluttering, occlusions, or any other geometrical problem.

So we have proposed a formalisation of an approach that combines a symbolic planner with a geometric task planner. GTP is more than a motion planner because it takes all the geometric decisions so HATP does not have to, which allows for more efficient combination since each planner reasons on what it is designed for. Thus during symbolic planning, every time an action is added its geometric counterpart is projected, or instantiated, in the geometry to ensure its feasibility in the current world state. Then GTP returns the effects of the action, on the geometry, in the form of shared literals that HATP uses to reason about the side effects of the actions in the plan. Hence combination allows to solve the ramification problem.

This combination of planners is capable of solving problem a purely symbolic planner can not

address. However, combining two search spaces makes the planning process quite slow and with too much possibilities to explore. In order to help both layers we proposed some improvements that would benefit of the hierarchical nature of HTN planning.

Indeed we have improved SGP with methods that are suited to the global view HTN encoding can offer on problems: the first consists in using constraints from the symbolic layer to guide the geometric search, the second provides the symbolic planner with heuristics calculated on the geometry and driving toward plans that are more likely to be geometrically feasible.

Using this complete system we were able to produce efficient solutions for HRI problems as well as a multi-UAV context, in the scope of the ARCAS project. However finding a solution is usually still slow so we proposed a better way to plan.

We have created a set of criteria that allow to sort the different backtrack points favouring to go faster toward the more promising solutions. Those criteria exploit knowledge from both symbolic and geometric layers: the reason about the causalities of actions, but also about the properties of the geometric actions, as well as other properties. This mechanism replaces the chronological backtrack that is usually used, changing a bit the depth first approach of the classical HTN planning process.

We wanted to further improve this search algorithm and made the proposal for a heuristic search to completely replace the depth first search. It would combine knowledge from both the symbolic and geometric communities: using a classical planner to evaluate the distance to the tasks goals, using a simplified geometric planner to obtain an estimate of the length of the geometric trajectories, and finally exploiting the criteria mechanism.

Limitations and Future Work

The first and most obvious future work concerns the new search algorithm and would be to complete its formalisation and implement it. In the thesis we proposed a set of features the test domains should exhibit to prove or disprove the efficiency of this approach, but new features and new domains should also be proposed.

About SGP, a limitation we have now is that the number of alternatives for each geometric action is fixed. It would be very interesting to study how to compute this number depending on both planners state and context. Maybe also taking inspiration from a process similar to the exploration limit presented in Lagriffoul et al. (2013), that would explore only certain geometric projections leaving the others for a post-process filling the gap.

Further extending this idea we could create a system that would do a first “virtual” decomposition of the HTN tree in order to determine the most likely decomposition without trying it in the geometry. And only then we could do the actual decomposition and instantiating the action in the geometry. This would allow to extract, in the virtual phase, constraints that could drive the search but also produce a first “skeleton” of the plan much faster than using our SGP approach.

Finally on SGP, a improvement that would be first useful to developers but that could bring new

results would be to add an ontology interface between the symbolic and geometric layer. Indeed when GTP sends the shared literals back there are just allocated to variables no transformation or inference is done. This new layer would be able to filter out invalid facts (a difference in the meaning for facts can occur since the two levels are not developed by the same communities), but it could also infer new ones, more complex and based on a computation that would use several facts.

GTP also has some limitations that should be parts of the future work, the main one is its inability to plan for several agents at the same time preventing any parallel plans. A workaround is to compute completely sequential plans and to parallelise them with a post process. But this approach is not satisfactory because the agents, by their actions are changing the environment: they move object, they can occlude trajectories that seemed feasible before and so on. Thus if GTP could plan for several agents in a shared environment we could compute multi-agent plans while ensuring their feasibility at both symbolic and geometric levels.

Finally another limitation occurs when GTP is asked for an alternative, it destroys any further action and do not reuse any part of the original action. It would be beneficial to have a way to exploit parts of previous computations in order to reduce a lot the search time. For instance the RRT tree from the original action could be used when computing an alternative solution: the initial and final pose may change but the search will have to roughly pass through the same space where we already have RRT points.

APPENDIX A

IMPROVING HATP IMPLEMENTATION

The goal of this appendix is to give more details on the implementation: the pseudo-code algorithm and some improvments conducted during this thesis. We start by presenting the algorithm with highlights corresponding to the integration with SGP. We proceed with information on how we enhanced the implementation and conclude with a section on possible future works.

A.1 Details on the implementation

This section presents the pseudo-code of HATP, our total-order HTN. The version presented here contains the modifications done to allow the integration with GTP but also the improvements compared to classical HTN: plan cost, multi-agent capabilities and social rules.

Algorithm 1 depicts the core function of HATP. This algorithm decomposes tasks until one of the stopping conditions is reached (Line 2): we asked to stopped at the first plan and it was found, the time limit was reached, or all the tasks were decomposed and there is no more plan left. The loop starts by getting the list of all the applicable tasks (Line 3), which are the ones whose preconditions hold true in the current state and with all their predecessors already in the solution plan (all causal links are satisfied). It is during this test that the actions with geometric counterpart are calling the geometric level to assess their feasibility. (Their effects and cost are used later.)

When the list of applicable tasks is updated, there are several cases. The simplest is an empty list with no other task to carry out, it signifies that all the tasks are decomposed and a plan was found, hence this plan can be returned. On the other hand the list can be empty but with some tasks left to do: some tasks are necessary but are not applicable so the plan is invalid, we discard it. Both when a plan is found or when the plan is invalid a backtrack is triggered to find a new plan. To do so we start by first choosing a point to backtrack to (Line 9), which in the classical implementation is the last

inserted (Last-In-First-Out strategy, or chronological order corresponding to deapth-first search) and then pick one of its alternatives.

When the list of applicable tasks is not empty, it can contain either one task or several. In the first case, when there is only one task, then it can be *applied* straight (Line 12, see later). Whereas if there are several applicable tasks, it implies that no ordering was given for those tasks: several orders are possible, and every possible linearisation will be tried. To try out all the different linearisations we create a backtrack point with as much backtrack alternatives as there are tasks. Each backtrack alternative contains a task to apply first (different in each backtrack point) and all other tasks are to be applied after: no order is specified for them. As a consequence we are creating an exponential number of backtrack alternatives but test all possible orders. In the code, Line 14 creates the backtrack point and return a single task representing the current backtrack alternative chosen among the newly created ones.

Algorithm 1 Simplified pseudo-code for our improved implementation of HTN

```
1:  function HTN-DECOMPOSITION
2:      while STOPPINGCONDITIONSNOTREACHED do
3:          App ← GETAPPLICABLETASKS
4:          if |App| = 0 then
5:              if ALLTASKSDECOMPOSED then
6:                  APPLYSOCIALCOSTONPLAN
7:                  return a plan                          ▷ A plan was found
8:              end if
9:              f_sel ← SELECTBACKTRACKPOINT
10:             BACKTRACKTO(f_sel)
11:         else if |App| = 1 then
12:             APPLYTASK(t), t ∈ App
13:         else
14:             t ← CREATEBACKTRACKALTERNATIVES
15:             APPLYTASK(t)
16:         end if
17:     end while
18: end function
```

Algorithm 2 shows the simplified code for the APPLYTASK function used in the first algorithm. It is strongly dependent on the type of the task to apply. If the task is an action it is directly added to the plan, its effects are used to update the symbolic state and its cost is added to the plan cost. However if the action has a geometric counterpart (lines with gray background) in addition to retrieving its effects and cost (to add up with the symbolic part), a backtrack point is created to allow to reconsider the decisions made by GTP. Since the actual choices are made over a continuous space we would

have to store an infinite number of backtrack alternatives as a consequence we decided to create a limited number of alternatives (this number is given by hand for now).

If the algorithm must apply a task that is an abstract task (or an HATP method) the planner must create backtrack points for the variable bindings and decomposition selection. We start by creating an alternative for each possible values for the variables (the values must be applicable, i.e. respect a set of predicates). We proceed by adding an alternative for each possible decomposition (whose variables are necessarily bound). Those two processes finish by adding the set of tasks from one of the alternatives to the set of all tasks to do. Finally we add the task to the current (partial) plan and go back to the first algorithm that will use the newly added tasks to decompose the rest of the plan.

Algorithm 2 Simplified code for the applyTask function

```
1:  function APPLYTASK(Task t)
2:      if t is an Action then
3:          eff ← ∅, cost ← 0
4:          if t has geometric counterpart then
5:              for i ∈ [1, maxNumberOfAlt] do
6:                  CREATEBACKTRACKINGALTERNATIVES
7:              end for
8:              eff ← GETGEOMETRICEFFECTS(t)
9:              cost ← GETGEOMETRICCOST(t)
10:         end if
11:         ADDTASKTOPLAN(t)
12:         UPDATEPLANCOST(t.cost + cost)
13:         UPDATESYMBOLICSTATE(t.effects ∪ eff)
14:     else
15:         for all variables v left free do
16:             for all acceptable values for v do
17:                 CREATEBACKTRACKALTERNATIVES
18:             end for
19:         end for
20:         for all decomposition d do
21:             CREATEBACKTRACKPOINTS
22:         end for
23:         ADDTASKTOPLAN(t)
24:     end if
25: end function
```

Note that the ADDTASKTOPLAN, used both for abstract or primitive tasks, is responsible for maintaining the causal links: it detects the last task that used a *resource* and create a link between this

task and the new one. A resource corresponds to an entity attribute: for instance if the new task to add has a predicate in its preconditions that tests a given entity attribute, the last action that changed it will have a causal link to the new task.

A.2 Improving the implementation

This section presents some improvements done on the implementation. The first part focuses on the grammar used in the domain parser, since it was incorrect on the work anterior to this thesis. We proceed by showing some modifications carried out only on the actual code. Then we give some details on the work done to facilitate its distribution and usage. Finally we conclude with a part on the future work.

A.2.1 Implementing a formal grammar

As shown in Section 3.4.2 HATP was already used in some HRI experiments however it had some limitations in the expressiveness of its language due to an improper grammar for the preconditions: for instance it was not possible to encapsulate FORALL. This section presents the correct grammar and shows some example of its usage.

A.2.1.1 Correct precondition grammar

The most fundamental element is the $term$: it represents a single element. It can be an entity e, an attribute (from an entity), an evaluable predicate $f(\dots)$ or a constant value. Additionally we can get the size of a set attribute.

$$term := e|(e,a)|size(e,a)|f(x_1,\dots,x_n)|const$$

Note that the notation for an attribute is (e,a) while in the actual language it is written e.a. The actual language allows nested attributes, which would be written e.a1.a2.a3 and would be accepted if e.a1 is an entity f with an attribute a2 who in turn must be an entity g with an attribute a3. To simplify the grammar here we represent a nested attribute as $(g.e3)$.

Combining terms allows us to create predicates, equivalent to a binary function. Again to simplify the grammar, we do not use the actual symbols from the language: » becomes $\in$, and « becomes $\notin$. Thus the two last form of the predicate allow to test that a term is in (respectively not in) an entity set attribute.

$$\begin{aligned} predicate := {} & term = term|term \neq term|term > term|term \geq term| \\ & term < term|term \leq term|term \in (e,a)|term \notin (e,a) \end{aligned}$$

Finally, we can combine the predicate with different rules to for the preconditions of a task. We allow disjunctions but they must be explicitly stated. The conjunction is the default construct, which explain the right recursion of the $precond$ rule.

$$precond := precond \vee precond | \forall (e' \in t), precond \rightarrow precond |$$
$$\exists (e' \in t), precond \rightarrow precond | predicate | precond$$

The FORALL clause is defined as $\forall (e' \in t), precond \rightarrow precond$ in the grammar, while its actual form is FORALL(Type entity, {select}, {ensure}) which test that for all entities of type Type, and that respect the predicates of select, they match the ensure predicates. Similarly we can test the existence of an entity with the EXIST clause (written $\exists (e' \in t)$).

In order to optimize the entities manipulation during the planning phase we want the operations to be type consistent. It is not represented in this grammar but when two terms are compared they must have the same type: we can assess that an entity attribute (e, a) equals a constant if they are both of the same type (number, or boolean, or string). Some operators such as $size$, $\in$ require the attribute to the be a set. To control the type during the parsing phase allows to reject domains where the type consistency is not enforced, hence we are assured that no planning step can create an invalid state.

A.2.2 Code re-factoring

Another important part of the work done on the implementation was focused on improving the code quality. The aim was to make HATP both more stable, easier to extend, and usable for robotics experiments.

The first improvement was to allow to stop right after the first plan is found. Before we could either compute all plans or set a time limit. In many cases the domain encoding is written is such a way that the first plan found corresponds to a “good enough” plan, and the other decomposition are just meant to tackle different scenario and deviation. In those cases, most of the time the first plan is the best plan. Besides this mechanism is coherent with the principle behind HTN: we provide a heuristic through the domain definition.

When a domain is completely described HATP must compile it in order to use it. The goal was to optimise the inner structures, they are very straight forward to use and extend. We carried out some work on making the compilation process more convenient and faster such that the development of new domains is easier, which quite often rely on a trial-and-error method.

To further ease this debugging phase but also to help users understand HATP in general, some efforts were done to create more significant output. The pieces of information HATP gives are more relevant, and help to better understand its inner state. Additionally the graphical-interface hatpconsole was made more stable. Along this interface a new one, hatptester, was developed to help send planning request to HATP. Figure A.1 presents this new interface.

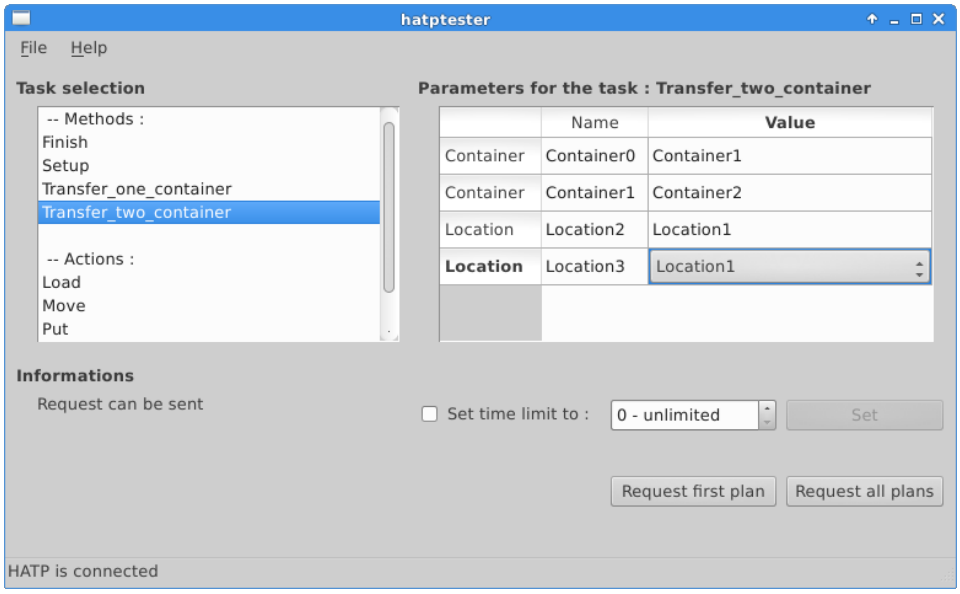


Figure A.1 – The graphical interface to send planning request, hatptester. It lists on the left all the tasks: the actions and methods. Once a task is selected it proposes all possible values for its parameters on the right. A time limit can be set, and finally it is possible to send a planning request for the first plan or for all plans.

A lot of efforts were dedicated to cleaning and documenting the code in general, there were many limitations and bugs that impacted the experiments. Most of them were solved and some new features were added, the most significant being the ability for HATP to run over the network, which allows to remotely control the planning process occurring on a robot: the different interfaces can be deported to a different computer with a display.

Cleaning the code also allowed to extend it, two different interfaces were created: on with GTP and another with the assembly planner presented in Section 6.1.3. Many new domains were created to use those interfaces, some of those domains were presented along this thesis.

Finally some work was devoted to share HATP and ease its learning. Next section presents a bit this work.

A.2.3 Share and teach HATP

The first work carried out to augment HATP visibility consisted in packaging the software in order for it to be very straight forward to install, hence making its distribution more convenient. HATP is indeed part of the robotpkg platform[1] which offers open-source tools for roboticists and provides a system similar to a package manager (detect dependencies between packages, update of outdated packages and so on).

[1] http://robotpkg.openrobots.org/

An online documentation assists users to install and use HATP. Some details are also available about its implementation so new developers can extend its functionalities. As mentioned before, this work was done in conjunction with code documentation and cleaning.

Since the code is open source, a code forge was set up offering access to the files but also providing some tools such as a bug tracker. Some work was dedicated to create code versions so it is easy to identify major stable versions.

In order to make HATP even more usable, some parts were ported to Windows, BSD and most Unix-like system allowing multi-platform execution. More recent works were done to integrate in the ROS framework, although this is incomplete.

A.2.4 Future work

Some of the future work has already been identified, two features that would be a good improvement to the language are: (1) add support for enumerations and (2) allow class inheritance. The former would allow to define the complete finite set of possible values for entities attributes. This would permit an even stronger type-consistency check during the domain-compilation phase: we could check that the value allocated to an attribute is valid, and so on. In general this would make the language more clear since the solutions now consist in defining values by hand which is error prone.

The second enhancement is to allow type inheritance in domain descriptions but it requires an in-depth proof of its validity. The motivation behind this is to offer the possibility to define the elements of the environment as precisely as possible while keeping the similar attributes shared so we can reason at both abstract and precise level. For instance, a common limitation appears when we want the robot to place an object on “something”, usually we place on a surface but if we want to place on another object (*Stack* action for instance) then we must define a general GeneralObject class, only then we can define the isOn attribute using this new class. However we need to add a type attribute to discriminate every time an entity is used to know if it is a surface or an object. The problem is that all the parameters specific to a type are encoded along the attributes for the other, for instance the GeneralObject can have an attribute reachableBy corresponding to the reach of the agents when it is an actual object, while it can also have an attribute store to register all the real objects placed on a surface. Besides this problem already exists with the Agent type, since most of the time we want to differentiate the humans and the robots.

We could solve those sorts of problems by defining a general common class that would have the shared parameters, then define classes that would inherit from the first and adding their own specific parameters, so each entity would have several types. This would allow to manipulate the entities both using their particular type and their inherited types.

Source code stats: Number of lines=77570 – Number of git commits=406 – Number of line edits=114673

APPENDIX B

DWR DOMAIN IN HATP LANGUAGE

This appendix presents the complete HATP domain description of the DWR domain. The domain consists of a robot that must move two containers from their original location to a goal location. Figure 3.1 presents the problem and the Section 3.2 describes it in more details.

```
1  factdatabase {
2      //Agent is already declared
3      define entityType Location;
4      define entityType Pile;
5      define entityType Container;
6
7      define entityAttributes Agent {
8          static atom string type;
9          dynamic atom Location at;
10         dynamic atom Container carry;
11     }
12
13     define entityAttributes Location {
14         static set Location adjacent;
15         dynamic atom bool occupied;
16     }
17
18     define entityAttributes Pile {
19         static atom Location attached;
20         dynamic set Container contains;
21         dynamic atom Container top;
22     }
23
24     define entityAttributes Container {
25         dynamic atom Location in;
26         dynamic atom Container on;
27     }
28
29     //Creations
30     R = new Agent;
31     K1 = new Agent;
32     K2 = new Agent;
33     L1 = new Location;
34     L2 = new Location;
35     P11 = new Pile;
36     P12 = new Pile;
```

```
37      P21 = new Pile;
38      P22 = new Pile;
39      C1 = new Container;
40      C2 = new Container;
41
42      //Initialisations
43      R.type = "ROBOT";
44      R.at = L1;
45
46      K1.type = "CRANE";
47      K1.at = L1;
48      K2.type = "CRANE";
49      K2.at = L2;
50
51      L1.adjacent <<= L2;
52      L1.occupied = true;
53      L2.adjacent <<= L1;
54      L2.occupied = false;
55
56      P11.attached = L1;
57      P11.contains <<= C1;
58      P11.top = C1;
59
60      P12.attached = L1;
61      P12.contains <<= C2;
62      P12.top = C2;
63
64      P21.attached = L2;
65      P22.attached = L2;
66
67      C1.in = L1;
68      C2.in = L1;
69  }
70
71  HTN {
72      action Move(Agent R, Location From, Location To) {
73          preconditions {
74              R.type == "ROBOT";
75              R.at == From;
76              To >> From.adjacent;
77              To.occupied == false;
78          };
79          effects{
80              R.at = To;
81              From.occupied = false;
82              To.occupied = true;
83          };
84          cost{costFn(1)};
85          duration{durationFn(1, 1)};
86      }
87
88      action Load(Agent K, Agent R, Container C, Location L) {
89          preconditions {
90              K.type == "CRANE";
91              K.at == L;
92              K.carry == C;
93              R.type == "ROBOT";
94              R.at == L;
95              R.carry == NULL;
96          };
97          effects{
98              K.carry = NULL;
99              R.carry = C;
100         };
101         cost{costFn(1)};
102         duration{durationFn(2, 2)};
103     }
104
105     action Unload(Agent K, Agent R, Container C, Location L) {
106         preconditions {
```

```
107             K.type == "CRANE";
108             K.at == L;
109             K.carry == NULL;
110             R.type == "ROBOT";
111             R.at == L;
112             R.carry == C;
113         };
114         effects{
115             K.carry = C;
116             R.carry = NULL;
117         };
118         cost{costFn(1)};
119         duration{durationFn(2, 2)};
120     }
121
122     action Take(Agent K, Container C, Pile P, Location L) {
123         preconditions {
124             K.type == "CRANE";
125             K.at == L;
126             K.carry == NULL;
127             P.attached == L;
128             C >> P.contains;
129             P.top == C;
130         };
131         effects{
132             K.carry = C;
133             P.contains =>> C;
134             P.top = C.on;
135             C.in = NULL;
136             C.on = NULL;
137         };
138         cost{costFn(1)};
139         duration{durationFn(1, 1)};
140     }
141
142     action Put(Agent K, Container C, Pile P, Location L) {
143         preconditions {
144             K.type == "CRANE";
145             K.at == L;
146             K.carry == C;
147             P.attached == L;
148         };
149         effects{
150             K.carry = NULL;
151             C.in = L;
152             C.on = P.top;
153             P.contains <<= C;
154             P.top = C;
155         };
156         cost{costFn(1)};
157         duration{durationFn(1, 1)};
158     }
159
160     method Setup(Container C, Agent R, Location L) {
161         goal {R.carry == C;};
162         {
163             preconditions {
164                 EXIST(Pile P, {P.attached == L;}, {P.top == C;});
165                 EXIST(Agent K, {K.type == "CRANE";}, {K.at == L; K.carry == NULL;});
166                 R.type == "ROBOT";
167                 R.at == L;
168             };
169             subtasks {
170                 P = SELECT(Pile, {P.attached == L; P.top == C;});
171                 K = SELECT(Agent, {K.type == "CRANE"; K.at == L; K.carry == NULL;});
172                 1: Take(K, C, P, L);
173                 2: Load(K, R, C, L)>1;
174             };
175         }
```

```
176     }
177
178     method Finish(Container C, Agent R, Location L) {
179         goal {R.carry == NULL;};
180         {
181             preconditions {
182                 EXIST(Agent K, {K.type == "CRANE";}, {K.at == L; K.carry == NULL;});
183                 R.type == "ROBOT";
184                 R.at == L;
185                 R.carry == C;
186             };
187             subtasks {
188                 K = SELECT(Agent, {K.type == "CRANE"; K.at == L; K.carry == NULL;});
189                 P = SELECT(Pile, {P.attached == L;});
190                 1: Unload(K, R, C, L);
191                 2: Put(K, C, P, L)>1;
192             };
193         }
194     }
195
196     method Transfer_one_container(Container C, Location L1, Location L2, Agent R) {
197         goal {C.in == L2;};
198         {
199             preconditions {
200                 C.in == L1;
201                 R.type == "ROBOT";
202                 R.at == L1;
203             };
204             subtasks {
205                 1: Setup(C, R, L1);
206                 2: Move(R, L1, L2)>1;
207                 3: Finish(C, R, L2)>2;
208             };
209         }
210         {
211             preconditions {
212                 C.in == L1;
213                 R.type == "ROBOT";
214                 R.at != L1;
215             };
216             subtasks {
217                 From = SELECT(Location, {From == R.at;});
218                 1: Move(R, From, L1);
219                 2: Setup(C, R, L1)>1;
220                 3: Move(R, L1, L2)>2;
221                 4: Finish(C, R, L2)>3;
222             };
223         }
224     }
225
226     method Transfer_two_container(Container C1, Container C2, Location L1, Location L2) {
227         goal {FORALL(Container C, {}, {C.in == L2;});};
228         {
229             preconditions {
230                 C1.in == L1;
231                 C2.in == L1;
232                 EXIST(Agent R, {}, {R.type == "ROBOT";});
233             };
234             subtasks {
235                 Robot = SELECT(Agent, {Robot.type == "ROBOT";});
236
237                 1: Transfer_one_container(C1, L1, L2, Robot);
238                 2: Transfer_one_container(C2, L1, L2, Robot)>1;
239             };
240         }
241     }
242 }
```

Listing B.1 – Complete HATP domain to solve the DWR problem.

GLOSSARY

action

See operator. 10

decomposition

In HATP, a method has several decompositions each representing a set of subtasks. It corresponds to a single method in classical HTN. 20, 117

literal

A literal corresponds to a fact in a state, in preconditions it is an instantiated predicate (or grounded predicate). iii, 12, 34, 35, 117

method

In classical HTN, a method is a recipe for decomposing an abstract task, there can be several methods for a single abstract task.

In HATP, a method corresponds to an abstract task and it has several decompositions. 10, 20, 107, 117

operator

In classical planning an operator is a task an agent can carry out, when it is instantiated (or grounded) it is called an action. The terms operator and actions are used interchangeably in this thesis.

In HATP, a primitive task maps to a single operator, so the term primitive task is equivalent to operator and action when describing HATP domains. 9, 20, 117

predicate

In first order logic, it is an expression that can hold true or false when instantiated (see literal) in a state. A set of predicates usually form a precondition. 11, 117

shared literal

A literal computed by the geometric layer and sent to the symbolic layer. 35, 46, 65, 71

ACRONYMS

AI

The Artificial Intelligence is the academic field of study which studies how to create computers and computer software that are capable of intelligent behaviour. 8

DWR

The Dock-Worker Robot problem is a common AI planning problem, it involves robots moving containers from piles to piles with the help of crane robots. 21, 113

GTP

The Geometric Task Planner offers an abstraction to a motion planner and can compute shared literals. 46, 105

HATP

The Hierarchical Agent-based Task Planner (previously Human-Aware Task Planner) is our HTN planner. 16, 33, 105

HRI

The Human-Robot Interaction field studies how to improve the qualities of robots to behave in a fashion more suitable to work in collaboration with humans. 17, 83

HTN

An Hierarchical Task Network is a sets of tasks representing different levels of abstraction. 1, 8, 37, 105

PDDL

PDDL stands for Planning Domain Definition Language, and is a very common language to describe symbolic planning domains. See McDermott et al. (1998) for more information. 18

SGP

The Symbolic-Geometric Planning consists in planning and reasoning about symbolic task and motion trajectories. Therefore SGP can also refer to the integration of HATP and GTP. 34, 85, 105

UAV

The Unmanned Aerial Vehicles, commonly known as drones, are vehicles without a pilot inside and in the scope of robotics it refers to aerial robots. 83

BIBLIOGRAPHY

Alami, R., Chatila, R., Clodic, A., Fleury, S., Herrb, M., Montreuil, V., and Sisbot, E. a. (2006a).
Towards Human-Aware Cognitive Robots.
In *AAAI Workshop on Cognitive Robotics*, pages 9–16.
3.4.2, 3.5.1

Alami, R., Clodic, A., Montreuil, V., and Sisbot, E. (2006b).
Toward Human-Aware Robot Task Planning.
In *AAAI Spring Symposium To Boldly Go Where No Human-Robot Team Has Gone Before*, pages 39–46.
3.5.1

Alami, R., Gharbi, M., Vadant, B., Lallement, R., and Suarez, A. (2014).
On Human-aware Task and Motion Planning Abilities for a Teammate Robot.
In *RSS Workshop Human-Robot Collaboration for Industrial Manufacturing*.
3.5.1

Alami, R., Simeon, T., and Laumond, J.-P. (1989).
A Geometrical Approach to Planning Manipulation Tasks. The Case of Discrete Placements and Grasps.
4.1, 4.3.3

Alami, R., Warnier, M., Guitton, J., Lemaignan, S., and Sisbot, E. A. (2011).
When the robot considers the human...
In *International Symposium on Robotics Research*.
3.3.3, 3.5.1

Ali, M. (2012).
Contribution to Decisional Human-Robot Interaction : Towards Collaborative Robot Companions.
PhD thesis, Institut National des Sciences Appliquées de Toulouse.
3.5.1

Ali, M., Alili, S., Warnier, M., and Alami, R. (2009).
An Architecture Supporting Proactive Robot Companion Behavior.
In *Adaptive and Emergent Behaviour and Complex Systems*.

3.4.2, 3.5.1

Alili, S. (2011).
Interaction décisionnelle Homme-Robot : planification de tâche pour un robot interactif en environnement humain.
PhD thesis, Université Toulouse 3 Paul Sabatier.
3.5.1

Alili, S., Alami, R., and Montreuil, V. (2008).
A Task Planner for an Autonomous Social Robot.
In *Distributed Autonomous Robotic Systems*, pages 335–344. Springer.
3.3.2, 3.5.1

Alili, S., Pandey, A. K., Sisbot, E. A., and Alami, R. (2010).
Interleaving Symbolic and Geometric Reasoning for a Robotic Assistant.
In *ICAPS Workshop on Combining Action and Motion Planning*.
3.5.1, 4.3.1.3, 4.6

Alili, S., Warnier, M., Ali, M., and Alami, R. (2009).
Planning and Plan-execution for Human-Robot Cooperative Task Achievement.
In *International Conference on Automated Planning and Scheduling*.
3.3.2, 3.5.1

ARCAS Deliverable D6.4, A. (2015).
Deliverable D6.4 – Structure assembly planning.
Technical report.
6.1.3

Arkin, R. (1998).
Behavior-based Robotics.
Bradford book. MIT Press.
2.1.1

Barry, J., Kaelbling, L. P., and Lozano-Pérez, T. (2013).
A hierarchical approach to manipulation with diverse actions.
In *International Conference on Robotics and Automation*.
4.3.3, 4.2

Bidot, J., Karlsson, L., Lagriffoul, F., and Saffiotti, a. (2015).
Geometric backtracking for combined task and motion planning in robotic systems.
Artificial Intelligence.
4.3.1.3, 4.2, 5.1.2.4

Brooks, R. (1987).
Planning is just a way of avoiding figuring out what to do next.
Technical report.
2.1.1

Butterfill, S. A. and Sebanz, N. (2011).
Editorial: Joint Action: What Is Shared?
Review of Philosophy and Psychology, 2(2):137–146.
6.2.1

Bäckström, C. and Nebel, B. (1993).
Complexity Results for SAS+ Planning.
Computational Intelligence, 11:625–655.
3.1.1

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., and Patoglu, V. (2009a).
Bridging the gap between high-level reasoning and low-level control.
Logic Programming and Nonmonotonic Reasoning, pages 342–354.
4.3.1.2, 4.4

Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., and Patoglu, V. (2009b).
From Discrete Task Plans to Continuous Trajectories.
In *ICAPS Workshop on Bridging The Gap Between Task And Motion Planning*.
4.3.1.2, 4.4

Cambon, S., Alami, R., and Gravot, F. (2009).
A Hybrid Approach to Intricate Motion, Manipulation and Task Planning.
International Journal of Robotics Research.
4.3.3, 4.6

Cambon, S., Gravot, F., and Alami, R. (2003).
Overview of aSyMov: Integrating motion, manipulation and task planning.
In *ICAPS Doctoral Consortium*.
4.3.3, 4.6

Cambon, S., Gravot, F., and Alami, R. (2004).
A Robot Task Planner that Merges Symbolic and Geometric Reasoning.
In *European Conference on Artificial Intelligence*, pages 895–899.
4.3.3, 4.6

Choi, J. and Amir, E. (2009).
Combining planning and motion planning.

In *International Conference on Robotics and Automation*, pages 238–244.
4.1, 4.3.2, 4.4

Clodic, A. (2007).
Supervision pour un robot interactif: Action et Interaction pour un robot autonome en environnement humain.
PhD thesis, Université Toulouse 3 Paul Sabatier.
3.5.1

Clodic, A., Alami, R., Montreuil, V., Li, S., Wrede, B., and Swadzba, A. (2007).
A study of interaction between dialog and decision for human-robot collaborative task achievement.
In *International Symposium on Robot and Human Interactive Communication*, pages 913–918.
3.3.3, 3.5.1

Clodic, A., Cao, H., Alili, S., Montreuil, V., Alami, R., and Chatila, R. (2009).
SHARY: a supervision system adapted to Human-Robot Interaction.
In *Experimental Robotics*, pages 229–238. Springer.
3.5.1

de Silva, L., Gharbi, M., Pandey, A. K., and Alami, R. (2014).
A New Approach to Combined Symbolic-Geometric Backtracking in the Context of Human-Robot Interaction.
In *International Conference on Robotics and Automation*.
3.5.1, 4.3.4, 4.6

de Silva, L., Lallement, R., and Alami, R. (2015).
The HATP Hierarchical Planner: Formalisation and an Initial Study of its Usability and Practicality.
In *International Conference on Intelligent Robots and Systems*, pages 6465–6472.
3.4.1, 3.5.1

de Silva, L., Pandey, A. K., and Alami, R. (2013a).
An interface for interleaved symbolic-geometric planning and backtracking.
In *International Conference on Intelligent Robots and Systems*, pages 232–239.
3.5.1, 4.3.1.3, 4.3.4, 4.6

de Silva, L. D., Pandey, A. K., Gharbi, M., and Alami, R. (2013b).
Towards Combining HTN Planning and Geometric Task Planning.
In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.
3.5.1, 4.6

Dearden, R. and Burbridge, C. (2013).

An Approach for Efficient Planning of Robotic Manipulation Tasks.
In *International Conference on Automated Planning and Scheduling*, pages 55–63.
4.3.1.2, 4.6

Devin, S. and Alami, R. (2016).
An Implemented Theory of Mind to Improve Human-Robot Shared Plans Execution.
In *International Conference on Human-Robot Interaction*.
3.4.2, 3.5.1

Dornhege, C., Eyerich, P., Keller, T., Brenner, M., and Nebel, B. (2010).
Integrating Task and Motion Planning Using Semantic Attachments.
In *AAAI Workshop on Bridging The Gap Between Task And Motion Planning*.
4.3.1.3, 4.4

Dornhege, C., Eyerich, P., Keller, T., Trüg, S., Brenner, M., and Nebel, B. (2012).
Semantic Attachments for Domain-Independent Planning Systems.
In *Towards Service Robots for Everyday Environments*, volume 76, pages 99–115. Springer.
4.3.1.3, 4.4

Dornhege, C., Gissler, M., Teschner, M., and Nebel, B. (2009).
Integrating symbolic and geometric planning for mobile manipulation.
In *International Workshop on Safety, Security and Rescue Robotics*.
4.3.1.3, 4.4

Dornhege, C., Hertle, A., and Nebel, B. (2013).
Lazy Evaluation and Subsumption Caching for Search-Based Integrated Task and Motion Planning.
In *IROS Workshop on AI-based Robotics*.
4.3.1.3, 4.4

Dovier, A., Formisano, A., and Pontelli, E. (2007).
Multivalued Action Languages with Constraints in CLP(FD).
In *International Conference on Logic Programming*, pages 255–270.
3.1.1

Dvorak, F., Toropila, D., and Bartak, R. (2013).
Towards AI Planning Efficiency: Finite-Domain State Variable Reformulation.
In *Symposium on Abstraction, Reformulation, and Approximation*.
3.1.1

Elkawkagy, M. and Bercher, P. (2012).
Improving Hierarchical Planning Performance by the Use of Landmarks.
In *AAAI Conference on Artificial Intelligence*.
5.2.3

Erdem, E., Haspalamutgil, K., Palaz, C., Patoglu, V., and Uras, T. (2011).
Combining High-Level Causal Reasoning with Low-Level Geometric Reasoning and Motion Planning for Robotic Manipulation.
In *International Conference on Robotics and Automation*, pages 4575–4581.
4.3.1.2, 4.4

Eyerich, P., Keller, T., and Nebel, B. (2009a).
Combining Action and Motion Planning via Semantic Attachments.
In *ICAPS Workshop on Combining Action and Motion Planning*.
4.4

Eyerich, P., Mattmüller, R., and Röger, G. (2009b).
Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning.
In *International Conference on Automated Planning and Scheduling*.
4.3.1.3

Ferrer-mestres, J., Frances, G., and Geffner, H. (2015).
Planning with State Constraints and its Application to Combined Task and Motion Planning.
In *ICAPS Workshop on Planning and Robotics*.
4.3.1.3, 4.6

Fikes, R. E. and Nilsson, N. J. (1971).
STRIPS: A new approach to the application of theorem proving to problem solving.
Artificial Intelligence, 2(3–4):189–208.
2.1.2

Fiore, M., Clodic, A., and Alami, R. (2015).
On Planning and Task Achievement Modalities for Human-Robot Collaboration.
In *Experimental Robotics*, pages 293–306. Springer.
3.5.1

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2014a).
FFRob: An efficient heuristic for task and motion planning.
In *International Workshop on the Algorithmic Foundations of Robotics*.
4.3.2, 4.2

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2014b).
Heuristic Search for Task and Motion Planning.
In *ICAPS Workshop on Planning and Robotics*, pages 148–156.
4.3.2, 4.2

Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2015).

Backward-Forward Search for Manipulation Planning.
In *International Conference on Intelligent Robots and Systems*, pages 6366—-6373.
4.3.2, 4.2

Gaschler, A., Kessler, I., Petrick, R. P. A., and Knoll, A. (2015).
Extending the Knowledge of Volumes Approach to Robot Task Planning with Efficient Geometric Predicates.
In *International Conference on Robotics and Automation*.
4.3.1.3, 4.4

Gaschler, A., Petrick, R. P. a., Giuliani, M., Rickert, M., and Knoll, A. (2013a).
KVP: A knowledge of volumes approach to robot task planning.
In *International Conference on Intelligent Robots and Systems*, pages 202–208.
4.3.1.3, 4.4

Gaschler, A., Petrick, R. P. a., Kr, T., Khatib, O., and Knoll, A. (2013b).
Robot Task and Motion Planning with Sets of Convex Polyhedra.
In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.
4.3.1.3, 4.4

Ghallab, M., Nau, D., and Traverso, P. (2004).
Automated Planning: Theory and Practice.
Morgan Kaufmann Publishers Inc.
2, 2.1.1, 2.2.1, 3.2

Ghallab, M., Nau, D., and Traverso, P. (2016).
Automated Planning and Acting.
Cambridge University Press.
Note: In Press.
2.1.2

Gharbi, M. (2015).
Geometric reasoning and planning in the context of human robot interaction.
PhD thesis, Université Toulouse 3 Paul Sabatier.
4

Gharbi, M., Lallement, R., and Alami, R. (2015).
Combining Symbolic and Geometric Planning to Synthesize Human-Aware Plans: Toward More Efficient Combined Search.
In *International Conference on Intelligent Robots and Systems*.
3.5.1, 1, 4.3.4, 4.6

Gibson, J. J. (1977).
"The theory of affordances", in Perceiving, Acting, and Knowing. Towards an Ecological Psychology.
Number eds Shaw R., Bransford J. Hoboken, NJ: John Wiley & Sons Inc.
4.4.1

Gravot, F., Cambon, S., and Alami, R. (2003).
aSyMov:A Planner That Deals with Intricate Symbolic and Geometric Problems.
In *International Symposium on Robotics Research*.
4.3.3, 4.6

Guitton, J. and Farges, J.-L. (2009a).
Taking Into Account Geometric Constraints for Task-oriented Motion Planning.
In *ICAPS Workshop on Bridging The Gap Between Task And Motion Planning*, pages 26–33.
4.3.1.3, 4.4

Guitton, J. and Farges, J.-L. (2009b).
Towards a Hybridization of Task and Motion Planning for Robotic Architectures.
In *IJCAI Workshop on Hybrid Control of Autonomous Systems*, pages 21–24.
4.1, 4.3.1.3, 4.4

Guitton, J., Warnier, M., and Alami, R. (2012a).
Belief Management for HRI Planning.
In *ECAI Workshop on Belief change, Non-monotonic reasoning and Conflict Resolution*, pages 27–33.
3.3.3, 3.5.1

Guitton, J., Warnier, M., and Alami, R. (2012b).
Vers une gestion des croyances pour la planification Homme - Robot.
Journées Francophones sur la Planification, la Décision et l'Apprentissage pour le contrôle des systèmes.
3.5.1

Haspalamutgil, K., Palaz, C., and Uras, T. (2010).
A Tight Integration of Task Planning and Motion Planning in an Execution Monitoring Framework.
In *AAAI Workshop on Bridging The Gap Between Task And Motion Planning*.
4.3.1.2, 4.4

Hauser, K. (2010).
Task Planning with Continuous Actions and Nondeterministic Motion Planning Queries.
In *AAAI Workshop on Bridging The Gap Between Task And Motion Planning*.
4.3.3, 4.6

Hauser, K. and Latombe, J.-C. (2009).
Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries.
In *International Conference on Automated Planning and Scheduling*.
4.1, 4.3.3, 4.6

Havur, G., Haspalamutgil, K., Palaz, C., Erdem, E., and Patoglu, V. (2013).
A case study on the Tower of Hanoi challenge: Representation, reasoning and execution.
In *International Conference on Robotics and Automation*, pages 4552–4559.
4.3.1.2, 4.4

Helmert, M. (2009).
Concise finite-domain representations for PDDL planning tasks.
Artificial Intelligence, 173(5–6):503–535.
3.1.1

Hertle, A., Dornhege, C., Keller, T., and Nebel, B. (2012).
Planning with Semantic Attachments: An Object-Oriented View.
In *European Conference on Artificial Intelligence*.
4.3.1.3, 4.4

Hoffmann, J. and Nebel, B. (2001).
The FF Planning System: Fast Plan Generation Through Heuristic Search.
Journal of Artificial Intelligence Research, 14:253–302.
2.1.2, 4.3.1.3

Kaelbling, L. P. (2011).
Planning in the Know : Hierarchical belief-space task and motion planning.
In *International Conference on Robotics and Automation*.
4.3.1.3, 4.2

Kaelbling, L. P. and Lozano-Pérez, T. (2011).
Hierarchical task and motion planning in the now.
In *International Conference on Robotics and Automation*, pages 1470–1477.
4.3.1.3, 4.2

Kaelbling, L. P. and Lozano-Pérez, T. (2013).
Integrated task and motion planning in belief space.
International Journal of Robotics Research, 32:1194–1227.
4.3.1.3, 4.2

Karlsson, L., Bidot, J., Lagriffoul, F., Saffiotti, A., Hillenbrand, U., and Schmidt, F. (2012).
Combining Task and Path Planning for a Humanoid Two-arm Robotic System.

In *ICAPS Workshop on Combining Task and Motion Planning for Real-World Applications*.
4.3.1.3, 4.2

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996).
Probabilistic roadmaps for path planning in high-dimensionalconfiguration spaces.
Robotics and Automation, 12(4):566–580.
4.3.3

Kelly, J.-P., botea, A., and Koenig, S. (2007).
Planning with Hierarchical Task Networks in Video Games.
In *ICAPS Workshop on Planning in Games*.
2.1.3

Lagriffoul, F. (2013).
Delegating Geometric Reasoning to the Task Planner.
In *ICAPS Workshop on Planning and Robotics*, pages 54–59.
4.3.1.1, 4.2

Lagriffoul, F., Dimitrov, D., Bidot, J., Saffiotti, A., and Karlsson, L. (2014).
Efficiently combining task and motion planning using geometric constraints.
International Journal of Robotics Research, 33:1726–1747.
4.1, 4.3.1.2, 4.2

Lagriffoul, F., Dimitrov, D., Saffiotti, A., and Karlsson, L. (2012).
Constraint Propagation on Interval Bounds for Dealing with Geometric Backtracking.
In *International Conference on Intelligent Robots and Systems*, pages 957–964.
4.3.1.2, 4.2

Lagriffoul, F., Karlsson, L., Bidot, J., and Saffiotti, A. (2013).
Combining Task and Motion Planning is Not Always a Good Idea.
In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.
4.3.4, 6.2.6

Lallement, R., de Silva, L., and Alami, R. (2014).
HATP: An HTN Planner for Robotics.
In *ICAPS Workshop on Planning and Robotics*, pages 20–27.
3.5.1, 4.3.4, 4.6

Lallée, S., Hamann, K., Steinwender, J., Warneken, F., Martinez-Hernandez, U., Barron-Gonzalez, H., Pattacini, U., Gori, I., Petit, M., Metta, G., Verschure, P., and Dominey, P. F. (2013).
Cooperative human robot interaction systems: IV. Communication of shared plans with humans using gaze and speech.

In *International Conference on Intelligent Robots and Systems*, pages 129–136.
6.2.4

Latombe, J.-C. (1991).
Robot Motion Planning.
Springer.
4.3.3

LaValle, S. (2006).
Planning Algorithms.
Cambridge University Press.
4.3.2

Lozano-Pérez, T. and Kaelbling, L. P. (2013).
A constraint-based method for solving sequential manipulation planning problems.
In *International Conference on Intelligent Robots and Systems*.
4.1, 4.3.1.2, 4.2

Mainprice, J., Sisbot, E. A., Jaillet, L., Corés, J., Alami, R., and Siméon, T. (2011).
Planning human-aware motions using a sampling-based costmap planner.
In *International Conference on Robotics and Automation*, pages 5012–5017.
4.4.4.3

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998).
PDDL – The Planning Domain Definition Language.
Technical Report, Yale Center for Computational Vision and Control.
B

Milliez, G., Lallement, R., Fiore, M., and Alami, R. (2015).
Using Human Knowledge Awareness to Adapt Collaborative Plan Generation, Explanation and Monitoring.
In *International Conference on Human-Robot Interaction*.
3.5.1, 6.2.6

Milliez, G., Warnier, M., Clodic, A., and Alami, R. (2014).
A framework for endowing interactive robot with reasoning capabilities about perspective-taking and belief management.
In *International Symposium on Robot and Human Interactive Communication*.
6.2.2.1

Montreuil, V. (2008).

Interaction Décisionnelle homme-robot: la planification de tâches au service de la sociabilité du robot.
PhD thesis, Université Toulouse 3 Paul Sabatier.
3.5.1

Montreuil, V., Clodic, A., Ransan, M., and Alami, R. (2007).
Planning human centered robot activities.
In *International Conference on Systems, Man and Cybernetics*, pages 2618–2623.
3.5.1

Morisset, B. and Ghallab, M. (2002).
Synthesis of supervision policies for robust sensory-motor behaviors.
In *International Conference on Intelligent Autonomous Systems*, pages 236–243.
2.1.3

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003).
SHOP2: An HTN Planning System.
Journal of Artificial Intelligence Research, 20(1):379–404.
2.1.3

Nau, D., Muñoz-Avila, H., Cao, Y., Lotem, A., and Mitchel, S. (2001).
Total-Order Planning with Partially Ordered Subtasks.
In *International Joint Conference on Artificial Intelligence*.
2.1.3

Nedunuri, S., Prabhu, S., Moll, M., Chaudhuri, S., and Kavraki, L. E. (2014).
SMT-Based Synthesis of Integrated Task and Motion Plans from Plan Outlines.
In *International Conference on Robotics and Automation*.
4.1, 4.3.2, 4.2

Pandey, A. K., Ali, M., Warnier, M., and Alami, R. (2011).
Towards Multi-State Visuo-Spatial Reasoning based Proactive Human-Robot Interaction.
In *International Conference on Advanced Robotics*, pages 143–149.
3.5.1

Plaku, E. (2012a).
Planning in Discrete and Continuous Spaces: From LTL Tasks to Robot Motions.
In *Advances in Autonomous Robotics*, volume 7429, pages 331–342. Springer.
4.3.2, 4.4

Plaku, E. (2012b).
Planning Robot Motions to Satisfy Linear Temporal Logic, Geometric, and Differential Constraints.

In *ICAPS Workshop on Combining Task and Motion Planning for Real-World Applications*, pages 21–28.
4.3.2, 4.4

Plaku, E. and Hager, G. D. (2010).
Sampling-based motion and symbolic action planning with geometric and differential constraints.
In *International Conference on Robotics and Automation*, pages 5002–5008.
4.3.2, 4.4

Renaudo, E., Devin, S., Girard, B., Chatila, R., Alami, R., Khamassi, M., and Clodic, A. (2015).
Learning to interact with humans using goal-directed and habitual behaviors.
In *RO-MAN Workshop on Learning for Human-Robot Collaboration*.
3.5.1

Sacerdoti, E. D. (1974).
Planning in a hierarchy of abstraction spaces.
Artificial Intelligence, 5(2):115–135.
2.1.2

Schüller, P., Patoglu, V., and Erdem, E. (2013).
Levels of Integration between Low-Level Reasoning and Task Planning.
In *AAAI Workshop on Intelligent Robotic Systems*, pages 73–78.
4.3

Shivashankar, V., Alford, R., Kuter, U., and Nau, D. (2013).
The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning.
In *International Joint Conference on Artificial Intelligence*, pages 2380–2386.
2.1.3, 4.3.1.3

Shivashankar, V., Kaipa, K. N., Nau, D. S., and Gupta, S. K. (2014).
Towards Integrating Hierarchical Goal Networks and Motion Planners to Support Planning for Human-Robot Teams.
In *International Conference on Intelligent Robots and Systems*.
4.3.1.3, 4.6

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014).
Combined Task and Motion Planning Through an Extensible Planner-Independent Interface Layer.
In *International Conference on Robotics and Automation*, pages 639–646.
4.3.1.2, 4.2

Srivastava, S., Riano, L., Russell, S., and Abbeel, P. (2013).

Using Classical Planners for Tasks with Continuous Operators in Robotics.
In *ICAPS Workshop on Planning and Robotics*.
4.3.1.2, 4.2

Tate, A. (1977).
Generating Project Networks.
In *International Joint Conference on Artificial Intelligence*, pages 888–893.
2.1.2

Tomasello, M. (2011).
Why We Cooperate.
Economics and Philosophy, 27:183–190.
6.2.1

Tomasello, M., Carpenter, M., Call, J., Behne, T., and Moll, H. (2005).
Understanding and sharing intentions: The origins of cultural cognition.
Behavioral and Brain Sciences, 28:675–691.
6.2.1, 6.2.4

Warneken, F., Chen, F., and Tomasello, M. (2006).
Cooperative activities in young children and chimpanzees.
Child Development, pages 640–663.
6.2.1

Warneken, F. and Tomasello, M. (2007).
Helping and Cooperation at 14 Months of Age.
Infancy, 11(3):271–294.
6.2.1

Warnier, M. (2012).
Gestion des croyances de l'homme et du robot et architecture pour la planification et le contrôle de la tâche collaborative homme-robot.
PhD thesis, Institut National des Sciences Appliquées de Toulouse.
3.5.1

Warnier, M., Guitton, J., Lemaignan, S., and Alami, R. (2012).
When the robot puts itself in your shoes. Managing and exploiting human and robot beliefs.
In *International Symposium on Robot and Human Interactive Communication*, pages 948–954.
3.3.3, 3.5.1

Wolfe, J. and Marthi, B. (2010).
Combined task and motion planning for mobile manipulation.

In *International Conference on Automated Planning and Scheduling*, pages 245–257.
4.3.1.3, 4.6

Wolfe, J., Marthi, B., and Russell, S. (2010).
Combined Task and Motion Planning for Mobile Manipulation.
Technical report, University of California at Berkeley, Department of Electrical Engineering and Computer Sciences.
4.3.1.3, 4.6

Zickler, S. and Veloso, M. (2009).
Efficient Physics-Based Planning: Sampling Search Via Non-Deterministic Tactics and Skills.
In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 27–34.
4.3.2, 4.4

Şucan, I. A. and Kavraki, L. E. (2011).
Mobile manipulation: Encoding motion planning options using task motion multigraphs.
In *International Conference on Robotics and Automation*, pages 5493–5498.
4.3.1.1, 4.2

Şucan, I. A. and Kavraki, L. E. (2012).
Accounting for uncertainty in simultaneous task and motion planning using task motion multigraphs.
In *International Conference on Robotics and Automation*, pages 4822–4828.
4.1, 4.3.1.1, 4.2