GEOMETRIC AND SYMBOLIC REASONING FOR MOBILE ROBOTICS

Julien Guitton and Jean-loup Farges

ONERA DCSD/CD
2 avenue Edouard Belin BP 74025
31055 Toulouse Cedex 4

Abstract

Autonomous mobile robots such as planetary rover need task and path planning abilities in order to fulfill their assigned missions. Task planning is caracterized by a symbolic reasoning and aims at defining a sequence of actions which will be executed to achieve the goals of the mission. Path planning allows to find some ways in the environment to reach these goals and corresponds to a geometric reasoning. Coupling these two kinds of reasoning presents open issues such as the description of the environment and the consideration of geometrical constraints that must be verified in order to act and move. This paper adresses these issues by studying different ways to couple a symbolic and a geometric reasoning and by proposing a hybrid planning architecture allowing to solve problems in which a mobile robot has to act and move in the environment.

Keywords

Autonomous mobile robot, task planning, path planning, agent reasoning, hybrid planning architecture.

1 INTRODUCTION

Autonomy of mobile robots is caracterized by the ability to act and move in their environment without human intervention. For applications with complex environment and goals, the agents autonomy implies symbolic and geometric reasoning. For instance, in order to accomplish its assigned mission, an autonomous planetary rover needs to plan all the actions it has to execute (eg. collect a sample of rock, take a picture) and to find a path between the different action areas.

This kind of problems was mainly addressed as orienteering problems [20] or in a multi-robot context as task allocation and path planning problems [18]. In the last case, a sequence of actions was usually pre-calculated and the goal turned out to be an assignment of each action to the different robots. In [2], a generic mission planner was developed. This planner tries to address mission problems by coupling a general purpose grammar interpreter with dynamic planning techniques. The grammar allows to define a set of tasks to be achieved. However, in this work, geometric constraints concerning the accomplishment of a task are not taken into account.

There is a need to investigate more specifically on the choice of actions (based on a symbolic description) that a robot will have to perform in order to fulfill its mission and on the inherent link between these actions and the movements needed to accomplish them.

We will see that a general purpose planner is not efficient for solving this kind of problem due to the difficulty associated with efficient and accurate representation of the environnement geometric concepts in its first order language. Therefore, there is a need to have different planners

interacting together. For instance, a task planner for the selection of the symbolic actions and a path planner in order to define the movements of the robot.

A naive approach is to define a hierarchy of planners. Once we have obtained a symbolic plan of actions, a specialized planner can calculate the movements of the agent. This approach raises the following questions [13]: (1) What kind of information should the path planner feed back to the task planner ? (2) How can the task planner use this information to generate a new plan ? In addition, with this method the sequence of actions must be fully calculated before realizing that it is not feasible in terms of movements. Another approach is to use a path planner during the action planning. This idea of relying on specific planners during planning [10] was used for the application with a mobile robot in [12]. In this work, the link between the generic planner and an itinerary planner was done using special symbolic attributes. However, the specialized planner builds its work on an accessibility graph and does not integrate reasoning about the geometric constraints of the problem. Moreover, the issue of dependency between subproblems of movement is not addressed. The other approaches dealing with the link between task planning and geometric reasoning are mainly approaches for one kind of specific problems [23]. In [3], an integrated planner is presented, in which the definition of states of the world contains symbolic and geometric informations. These informations are defined by the use of a set of types and specific predicates that allow to establish a link between symbolic planning and manipulation planning. Nevertheless, these works do not really show, in the one hand, the advantages of this hybrid approach in term of computation time and quality of the solution and, in the other hand, how to efficiently bind the main planner with the specialized reasoner in a general way.

This paper outlines some proposals allowing to bring together task planning and path planning in order, for a mobile robot, to accomplish its assigned mission. In the first part, we present the existing relationship between symbolic reasoning and geometric reasoning in the context of mobile robotics and make some comparisons of performances between the different ways of coupling these two kinds of reasoning. Next, we propose and detail an architecture allowing a cohabitation between a task planner and a specialized reasoner dedicated to the path computation. We also introduce the concepts we wish to implement within this architecture, and explain how to implement them at the different levels of the architecture.

2 TOWARDS A HYBRID PLANNING MODEL

The purpose of this part is to demonstrate how the use of a specialized reasoner for the path computation can drastically improve the performance of the planning process and to investigate the different possibilities to link the main planner with the specialized one. First of all, we introduce some definitions concerning planning in order to better understand the purpose of this article. Next, a comparison between a classical planning model and a hybrid model is done. Then, we propose and study different methods allowing to link a classical task planner with a specialized reasoner.

2.1 Introduction to planning

The word *planning* is a general term which can take several significances even in the artificial intelligence and robotics fields. Indeed as there are many types of actions, they are many types of planning among them :

- Task planning which allows to define the set of actions to execute in order to achieve the objectives ;
- Path planning which is necessary to allow the robot to move in its environment ;

- Perception planning which is used in some tasks such as environment modeling, object identification or localization ;
- Navigation planning which combine path planning and perception planning.

2.1.1 Task planning

Task planning is concerned by the reasoning about the necessary actions a agent has to achieve in order to realize its assigned mission. It is a symbolic reasoning and aims at constructing a plan of actions allowing, from an initial state, to reach a final state in which all the goals will be achieved. Transitions between states are done by applying the effects of the planning operators to the current state.

Task planners can be classified in three categories [8] : domain-specific, domain-independent and domain-configurable planners. The first are tailor-made for use in a given planning domain. They are powerful to solve the problems they are designed for but cannot be used in different domains. Domain-independant or classical planners are conceived to solve a large class of problems. They take as input a description of the domain related to the problem to solve. They are mainly used with benchmark domains. If there is no restriction on the kind of problems they can solve, they are restricted on classical planning domains, ie.domains satisfying a set of assumptions [15] : complete knowledge about a deterministic, static and finite system with restricted goals and implicit time. Even with these restrictions, for complex problems, the number of states of the search space can be too large to be solved in a realistic computation time. Domain-configurable planners are based on domain-independant planners but use domain-specific knowledge to constrain the size of the search space. This knowledge can be expressed in the domain description under the form of a Hierarchical Task Network (HTN) [5, 16] or control rules [1]. This class of planners run much faster than classical planners, nevertheless only certain types of knowledge can be capturated with these techniques [22].

In classical planning the aim is, starting from an initial state of the world and a set of goals, to reach a final state in which all the goals will be achieved. This final state is reached by applying successively operators from a set of operators representing all the possible actions to the current state of the world. The selection of an appropriate operator is done by testing its preconditions. Then, its effects are applied to the current state, thus corresponding to a finite state transition.

Definition 2.1 (Planning operator)
An operator is defined by a tuple o = <head(o), preconditions(o), effects(o)> where

- *head(o) is the head of the operator and is defined as* $n(x_1,...,x_k)$ *where n is its name and* $x_1,...,x_k$ *its parameters ;*
- *preconditions(o) is a list of predicates representing the necessary properties to be true in order to apply the operator*
- *effects(o) is a set of predicates to add (or remove) to the current state of the world when applying the operator.*

Definition 2.2 (Planning problem)
A problem is a tuple $\mathcal{P}$ *= <O,* s_0*, g> where*

- *O is the set of operators that can be applied ;*
- s_0 *is the initial state ;*
- *g is the set of goals to be achieved.*

In hierarchical planning, or HTN planning for Hierarchical Task Network planning [5] the objective is not to achieve a set of goals but to decompose a set of tasks. The planning domain contains a set of methods which specify the way to decompose a task into subtasks and a set of operators corresponding to the primitive tasks, ie. the non-decomposable tasks. These methods can be seen as recipes. A problem is described by an initial state and by a set of high-level tasks to decompose until obtaining primitive tasks corresponding to the actions an agent can perform in the environment.

Definition 2.3 (HTN Method)
An method is a tuple $m = < head(m), \{decomposition_1(m),...,decomposition_i(m)\} >$ *where :* $decomposition_i(m)$ $d_i = <preconditions(d_i), reduction(d_i)>$

- *head(m) is defined as* $n(x_1,...,x_k)$ *where n is the name of the method and* $x_1,...,x_k$ *its parameters ;*
- $decomposition_i(m)$ *is one of the possible decomposition of the method ;*
- *reduction(*d_i*) defines the sequence of tasks resulting from the decomposition of the method ;*
- *preconditions(*d_i*) is a list of predicates representing the necessary properties to be true in order to decompose m.*

In HTN planning a problem can be redefined as :

Definition 2.4 (HTN Planning problem)
A problem is a tuple $\mathcal{P}_{\mathcal{H}} = <O, \mathcal{M}, s_0, \{t_1,...,t_n\} >$. *where*

- *O is the set of operators that can be applied ;*
- $\mathcal{M}$ *is the set of methods ;*
- s_0 *is the initial state ;*
- $\{t_1,...,t_n\}$ *is the set of tasks to be decomposed.*

2.1.2 Path planning

Path planning aims at defining a path from an initial point representing the current agent location to a destination point while avoiding the obstacle and by satisfying the provided constraints (eg. kinematic constraints). The most well-known methods to plan a path are based on the construction of the environment representation (the configuration space) using a graph or a grid. These methods can be classified in three families : skeleton approaches, probabilistic approaches and cell decomposition approaches.

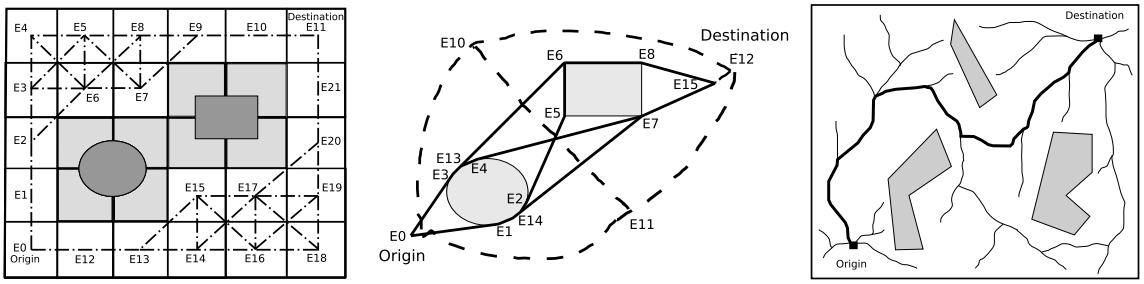
In the skeleton approaches the set of possible paths is reduced to a network of one-dimensional lines. The well-known skeletons are the visibility graph and the Voronoi diagram (figure 1(b)). The visibility graph is used when the solution path must be as shortest as possible. It connects between them the obstacle corners that are mutually visible. If the robot is required to stay away from obstacles, the Voronoi diagram is used. It is defined by the set of points that are equidistant from two ore more obstacles. Visibility graphs are often used in robotic applications. To be more realistic, the obstacles can be augmented with a Minkowski sum [19], ie. the robot cannot pass very close to the obstacles and so travels in a safety way.

The probabilistic approaches are efficient methods to compute paths for robot with a high degrees of freedom. Nevertheless, they are incomplete methods, ie. they don't guarantee to find a solution in a finite time even if it exists a possible path. The two methods that are the most

used are the Probabilistic RoadMap (PRM) [11] in which a graph is constructed by connecting randomly choosen configurations, and the Rapidly Random Tree method (RRT) [14] which is an improvement of the PRM method (figure 1(c)).

In the cell decomposition approaches the free space is decomposed into a set of cells and the adjancy relationships are computed (figure 1(a)). The simplest method is to decompose the environment into regular cells, but more complex decompositions can be adopted as, for instance, polygonal decomposition or quadtree decomposition.

After the construction of the configuration space using one of these approaches, one needs to find a solution path between an initial and a final position. The main used algorithms are the A* algorithm and the Dijkstra algorithm [4].



(a) Regular cell decomposition (b) Visibility graph and Voronoi diagram (c) Rapidly Random Trees

Figure 1: Three techniques for path planning

2.2 Classical planning versus hybrid planning

Nowadays, domain-independent planners become more and more efficient and are able to solve more and more complex problems. However, since they are not intended for a specific kind of problems, their designers try to make them more general they can in order to be able to compare them to other state-of-the-art planners. As a consequence, the problems used (the benchmarks) are often idealized or simplified versions of real problems [9].

The ability for these classical planners to be able to solve a wide range of problems with the same algorithmic has its drawbacks : some parts of complex problems would be solved more efficiently with a reasoner intended for this kind of subproblems. This is the case for path planning subproblems in which a mobile agent has to move in the environment.

We propose to make a comparison of performance between a planning approach in which a classical planner is used alone and a hybrid model in which the classical planner delegates the subproblems of searching a path to a path planner. In both approaches, a HTN planner similar to SHOP2 [16] is used. This comparison is done on the *Rover domain*.

2.2.1 Example problem

In the Rover domain, used to compare planners performances at the third International Planning Competition [21] and inspired by planetary rovers problems, a set of rovers must collect samples of rock and soil, take some pictures and communicate them to a fixed lander station. The rovers must travel in the environment in order to fulfill their missions. We use this planning domain to compare the classical planning approach, in which the movements are classical actions computed by the task planner, with the hybrid planning model in which the movements are delegated to a specialized reasoner.

The problem used with this domain is as follow : a single robot must do some soil samples on specified points. It has to move in the environment and avoid the obstacles. The environment represented in the figure 2 is composed of obstacles (in grey) and areas on which soil samples are required (in yellow). The green point represents the starting position of the robot.

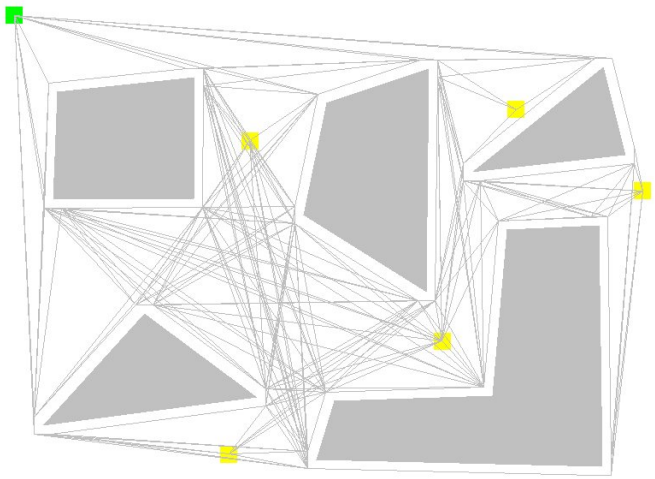


Figure 2: The environment map used with the rover domain and the associated visibility graph

The planning domain and the problem are written in the classical planning formalism using the PDDL language [6]. The initial state is composed of symbolic predicates modeling the rover capacities and resources. The environment is modeled with a set of predicates representing the waypoints and all the possible paths between these waypoints. These geometrical data are extracted from a visibility graph associated with the environment map (figure 2).

The vertices of the visibility graph are translated into symbolic waypoints and the set of edges represents all the possible paths between the waypoints. These paths are expressed with the symbolic predicate `can_traverse` as in the original version of the rover problem :

```
(waypoint waypoint0)(waypoint waypoint1)
(can_traverse rover1 waypoint0 waypoint1)
```

2.2.2 Classical planning model

In the classical approach, all the planning process is computed by the hierarchical task network planner .

The search for a physical path allowing the robot to accomplish its mission is done by trying to unify the preconditions of the operator `navigate` with the current state of the world.

```
(Operator (!navigate ?r ?x ?y)
 ;; preconditions
 ((rover ?r)(waypoint ?x)(waypoint ?y)
  (at ?r ?x)(can_traverse ?r ?x ?y))
 ;; effects
 ((not(at ?r ?x))(at ?r ?y))
)
```

The operator `navigate` is called by a method whose role is to avoid making loop by recording the waypoints which have already been traversed.

```
(Method (navigate ?rover ?from ?to)
  ;; the destination is reached
  ((at ?rover ?to))
  ()
  ;; the rover can go directly to the destination
  ((visible ?from ?to))
  ((!navigate ?rover ?from ?to))
  ;; in the other case, we choose a waypoint and mark it as visited
  ((waypoint ?mid)(visible ?from ?mid)(not (visited ?mid)))
  ((!navigate ?rover ?from ?mid)(!!visit ?mid)(navigate ?rover ?mid ?to)
   (!!unvisit ?mid))
)
```

2.2.3 Hybrid planning model

In the hybrid approach the path planning subproblems are delagated to a path planner instead of being computed by the task planner. The path planner implements the Dijkstra algorithm to find the shortest path in the visibility graph between two nodes (ie. two waypoints) given by the hierarchical task network planner.

The link between the HTN planner and this specialized reasoner is done through the use of geometrical preconditions. When the main planner meets this kind of preconditions, it sends to the path planner the coordinates of the starting and final points. If a path exists between these two points, the path planner replies with a positive answer and the main planning process can continue. Otherwise, the HTN planner backtracks and tries to plan another action.

```
(Operator (!navigate ?r ?x ?y)
 ;; preconditions
 ((rover ?r)(waypoint ?x)(waypoint ?y)
  (at ?r ?x)(posX ?x ?c1x)(posY ?x ?c1y)
  (posX ?y ?c2x)(posY ?y ?c2y))
 ;; geometrical data
 ((from ?c1x ?c1y)(to ?c2x ?c2y))
 ;; effects
 ((not(at ?r ?x))(at ?r ?y))
)
```

In this version of the rover domain, the initial state does not contain `can_traverse` predicates. However the waypoints coordinates have been added to the initial state in order to inform the path planner of the coordinates of the starting and sampling points :

```
(waypoint waypoint5)
(posX waypoint5 60)(posY waypoint5 99)
```

This addition was done by using information of the environment map before the planning process. As we use a static environment for our tests, the computation time to generate the initial state in this approach and in the classical approach was not taken into account.

2.2.4 Comparison between the classical model and the hybrid model

The comparison between the classical model and the hybrid model is done on the rover domain. We run each approach on five examples having an increasing difficulty. In the first example, the rover has to accomplish a single objective. In each following example, a new objectif is

added. All the objectives are identical, that is to make a sample of soil at a specified location. We compare the computation time spend by each method on the five examples to find a solution plan as well as the quality of the solution in term of traveled distance.

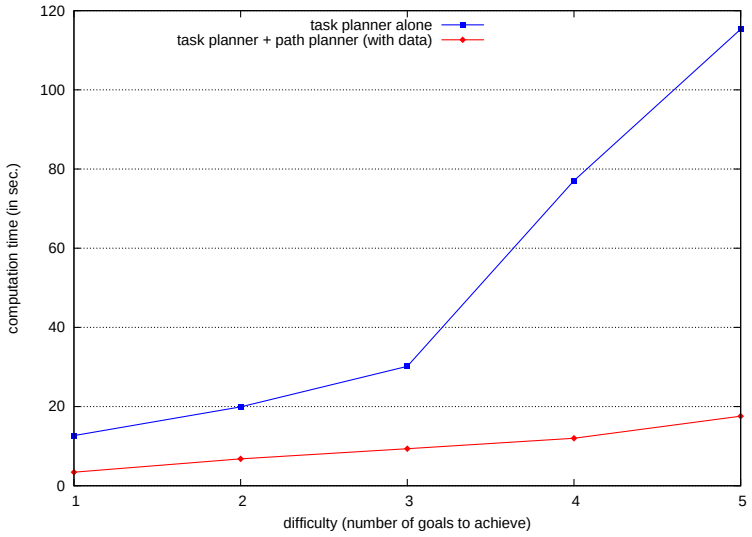


Figure 3: Comparison of the computation time for the two approaches

The computation time[1] to find a solution plan for each problem is shown on the figure 3. The time spend by the hybrid model is lowest and increases lineary with the difficulty, whereas with the classical approach, the computation time increases much more faster. This is due to the fact that the search for a path between the two points is not guided. Indeed, the HTN planner tries to find a possible path in the same way it does for classical task, ie. by decomposing tasks into subtasks until obtaining primitive actions in which preconditions must be unified with the current state of the world.

The comparison of the traveled distance by the rover in order to fulfill its mission for each case is represented in the figure 4. The traveled distances with the hybrid planning model can be used to judge the quality of the paths found with the classical approach. Indeed, as the path planner used in the hybrid approach is based on the Dijkstra algorithm, the solution path found between two objectives is optimal in terms of length. In the classical framework, the more the rover has to move, the less the distance covered is optimal. These results corroborate the results obtained with the computation time comparison : in the classical approach, the subproblems of finding paths allowing to navigate between the objectives are not easily tackled by the task planner.

In the International Planning Competition of 2002, the competitors presenting the SHOP2 planner use external calls to specialized Lisp function for computing the best paths. This easy way to improve the performance of the planner can be seen as the use of a hybrid planning model in which some parts of the problem are treated by specific algorithms.

2.3 Linking the two planners

The use of a specialized planner for some identified subproblems of the planning problem can reduce the resolution time and increase the quality of the solution. The question, now, is how to efficiently link the main symbolic planner with a reasoner adapted to the kind of subproblems present in the planning problem. We propose to study different methods to link the task planner with a path planner in order to solve path planning subproblems.

[1]The experiments were done on a SunBlade 1500 workstation. The task planner and the path planner have been developed using Java 1.5

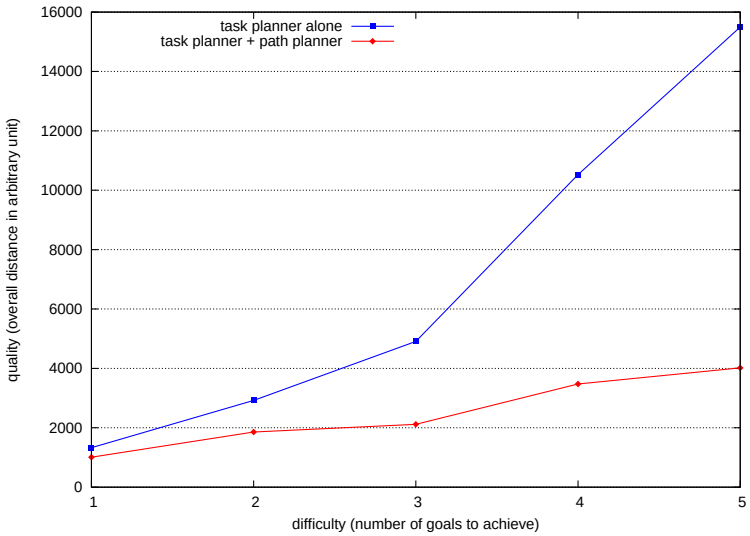


Figure 4: Comparison of the plan quality in term of traveled distance for the two approaches

2.3.1 Different methods to link the task planner and the path planner

We can consider different ways to bind the main planner and the specialized reasoner. A naive approach is to define a hierarchy of planners. This hierarchical approach is the common approach used for the design of robotic architecture [13]. Indeed, most of the time a symbolic plan is computed first and thereafter this abstract plan is refined, ie. a geometrically feasible path satisfying the symbolic *navigate* actions of the plan is looked for [7]. If there is no path satisfying the constraints, the task planner will try to find another solution plan.

This approach can be improved by recording the actions which produce geometrical constraints. If the path planner does not find a solution, the backtrack process is carried out to the infeasible action.

Another possible solution to link the main planner with the path planner is to interleave their execution. Geometrical constraints are sent to the path planner during the main planning process each time they are encountered. If the path planner finds a solution allowing to move the rover, it sends back a positive answer. If the geometrical constraints are unsatisfiable it returns a failure. The calls of the specialized planner can be seen as precondition tests.

2.3.2 Comparison between the three methods

These three approaches (hierarchical, hierarchical with backtrack and interleaved executions) are tested on five examples in which a rover must take pictures of four objectives. The rover has to move in the environment in order to fullfill its assigned goals. For each goal two points of action are defined, but the first one, corresponding to the achievement area of the action that the HTN planner tries to plan in first, is possibly inaccessible (e.g. the point is defined inside an obstacle). In the problem Pb1-0 all the first points are possible, whereas in the last problem (pb1-4), only the second point of each goal is accessible.

For all methods, a HTN planner is linked with a path planner based on a visibility graph and a Dijkstra algorithm. The architecture implemented for the three methods is identical except for the manner of transmitting the geometrical data. In the hierarchical methods, the geometrical requests are sent to the path planner at the end of the planning process, ie. when a symbolic solution plan is found. Whereas in the interleaved method they are transmitted during the process.

Figure 5 illustrates the comparison of the computation time between the three presented methods to link the main planner with the path planner on the five examples. As it was expected, using

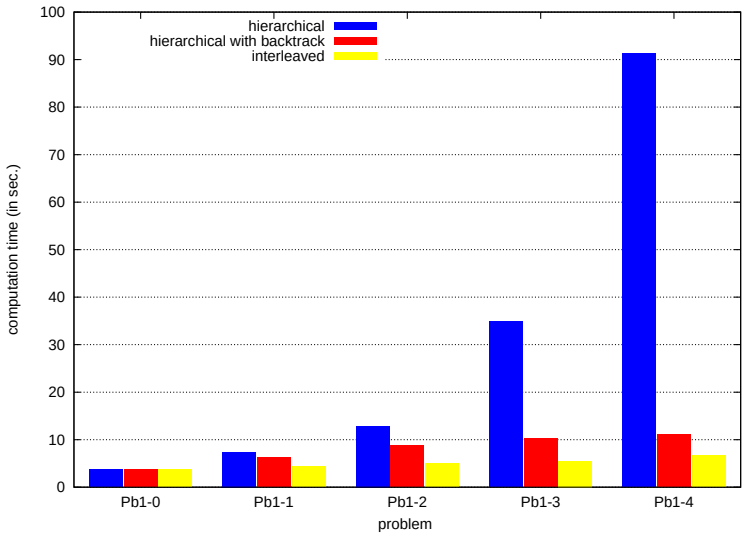


Figure 5: Comparison of the three hybrid methods

Problem	hierarchical	hierarchical with BT	interleaved
Pb1-0	4	4	4
Pb1-1	8	8	5
Pb1-2	17	11	6
Pb1-3	43	13	7
Pb1-4	112	14	8

Table 1: Comparison of the number of requests sent by the task planner to the path planner for each method

planning information for backtracking to the first unfeasible action is more efficient than using a blind hierarchical approach. The computation time for the method with interleaved executions of the task planner and the path planner is always lower than the computation time for the hierarchical methods. Indeed, if the current specified location is not reachable, the task planner can immediately backtrack and try to find another solution.

Table 1 resumes the number of requests sent by the main planner to the path planner for the five examples with each methods. The number of sent requests is correlate with the computation time. This correlation indicates a causal effect : the main reason for an increase of the computation time is an increased number of paths to be computed.

With the interleaved approach, the task planner will not try to plan the achievement of the next goal while a path to the current objective location is not found, ie. while the geometrical constraints are not satisfied. Whereas, with the hierarchical methods, a symbolic final plan is found before trying to satisfy the entire set of geometrical constraints. As a consequence, the same, possibly unsatisfiable, geometrical constraints are sent to the path planner until the backtracking process has not reached the geometrically unfeasible action.

2.4 Summary

In this part, we have shown that a hybrid planning model, in which the subproblems of finding a path in the environment is delegated to a specialized reasoner, has better performances than a classical planner in which all the problem is solved by the task planner. Moreover, coupling the task planner with the specialized reasoner using an interleaved approach is more efficient than the hierarchical approach.

3 IMPLEMENTATION OF THE HYBRID PLANNING ARCHITECTURE

The type of robotic problem we want to solve requires the use of a task planner and a path planner. The task planner aims at defining a sequence of actions to carry out the objectives of the mission. The path planner is used to define the movements of the robot. We propose an architecture in which task planning and path planning are interleaved.

The main proposal is as follow : the movements are not symbolic actions (ie.managed by the task planner) but are *geometric preconditions* to symbolic actions. For instance, in order to sample some rock the robot must be able to move to the relevant area. Depending on the type of robot, these preconditions may include only static geometric constraints or kinematic constraints. For instance, the turning radius may be a function of the speed of the robot.

The exchanges between the two planners are carried out by a set of *requests* sent from the task planner to the path planner. These requests specify the geometric and kinematic constraints which must be respected in order to plan an action.

In addition, the path planner can provide some *advices* to the task planner. These advices aim at re-scheduling the planned actions according to the specialized planner feedbacks. They aim at solving the global optimality problem concerning the agent movements by proposing alternatives to the current plan.

3.1 The planning architecture components

This part details the planning architecture of the robot deliberative module allowing to connect a general purpose planner and one or several specialized planners dedicated to the different mobile robots. For now, the architecture is composed of a task planner and only one specialized path planner. The two planners are connected through a communication interface (figure 6).

At the initialization phase, the prediction provides the informations needed by the architecture in order to solve the problem. These informations are the symbolic description of the possible actions the robot can do, the environment map and description, and the problem to solve. At the end of the processing, the interface sends the solution plan to the execution module. This solution plan contains the symbolic actions and the movement actions calculated by the two planners.

3.1.1 The task planner

The aim of the task planner is, starting from an initial state representing the state of the environment (in a symbolic formalism) and the state of the robot at the beginning of the mission, to reach a final state in which all the goals of the mission will be achieved. The achievable actions are expressed by a set of planning operators. The sequence of necessary actions to the achievement of an objective can be defined as a *recipe*. These recipes are expressed so as to be able to start them at any level during the execution, according to the current environment and robot state.

Thus, we use a task planning algorithm based on hierarchical planning techniques. Our hierarchical planning algorithm allows to express a set of partially ordered tasks as in Nau's work [17]. In the example below, the three tasks are independent and can be achieved in any order (thanks to the use of the keyword `unordered`).

```
(:unordered
  (film_objective rover1 locationA)
  (film_objective rover1 locationB)
  (sample_rock rover1 locationA)
)
```

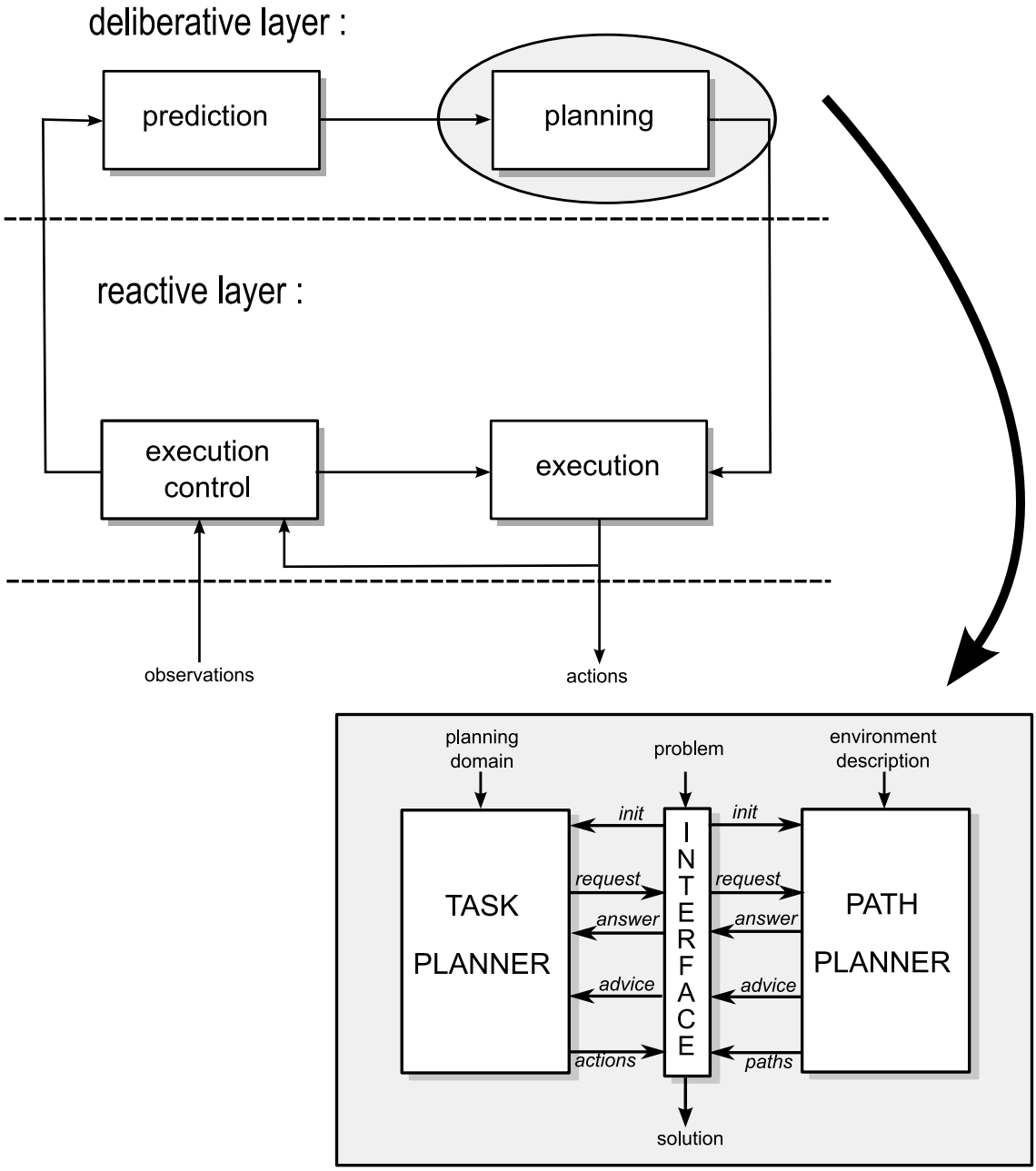


Figure 6: Description of the planning module

A problem for the task planner is expressed in the form of a planning domain and a planning problem. The domain is a set of operators (primitives tasks) and methods (high-level tasks). A problem is composed of the initial state which describes the initial symbolic state of the world and the goals in the form of a sequence of high-level tasks to decompose.

3.1.2 The path planner

The path planner must be able to provide a route between two points or two geographical areas specified by a set of geometric constraints. Various internal representations of the environment can be chosen : cells decomposition, use of a grid, representation with a visibility graph... However, this choice can affect the quality of the final solution in terms of movements of the robot. Moreover, it must respect the kinematic constraints of the robot (eg. its speed).

The goal of the path planner is to answer to the requests of the general purpose planner by indicating, on the one hand, if a possible path between the specified points exists and, on the other hand, if the robot can be moved during the action according to the specified constraints.

It must also be able to maintain an historic of previous movements in order to suggest to the task planner some optimizations as, for example, the inversion of two or more tasks in order to decrease the total traveled distance of the robot. Moreover, the search for a path can produce more results than the path itself. These results and the already computed paths can be re-used to optimize the entire path.

In our implementation of the path planner, the Dijkstra algorithm is used to find the shortest path in a visibility graph representing the environment.

3.1.3 Interface between the two planners

The interface between the general purpose planner and the specialized one allows a communication between them. They are not directly connected to permit the integration of a specialized planner corresponding to the kind of problem which will be treated. For instance, in the above example, a path planner can be used. Whereas, in a manipulation problem in which an articulated arm has to grasp objects, a motion planner will be more suitable.

This interface will also allow to deal with problems in a multi-robot context. In this kind of problem, a general purpose planner will be linked with several specialized planners in charge of the different robots.

The first purpose of the interface is to initialize the two planners. It receives data from the the mission management system of the robot and sends to the task planner the planning domain description, and to the path planner the environment description.

After the initialization phase, the interface aims at controlling the exchanges between the two planners. It transfers requests from the task planner to the corresponding specialized planner (in the case of a multi-agent architecture) and, according to the information provided by the path planner, it can propose some advices to the task planner.

3.2 Planning formalism

As the paths can be calculated by a specialized planner, the presence of the *navigate* operator is not necessary in the planning domain. Indeed, it can be assumed that movements are preconditions to the execution of an action. For example, in order to make a sample of soil at a specific location, the robot must travel to this location and then make the sample. In this case, the operator description can be augmented with the geometrical constraints needed to perform the action related to this operator in term of movements before and during the action.

Thus the planning operators include two new kind of preconditions called respectively *attitude preconditions* and *behaviour preconditions*. These preconditions allow the expression of a set of geometrical constraints which are necessary to the achievement of the action. Attitude preconditions are used to express the attitude the robot must have in order to realize an action, ie. its position, orientation... Behaviour preconditions define the behaviour of the robot in terms of movements during the achievement of the action. These conditions allow to take into account the movement of the robot in the case of durative action. For example, the operator `film_objective` can be described as follow :

```
(Operator (!film_objective ?r ?o)
  ;; symbolic preconditions
  ((rover ?r) (objective ?o)
   (camera ?c) (has_camera ?r ?c)
   (is_calibrated ?c))
```

```
  ;; attitude preconditions
  ((distance(r.pos, o.pos)>= 10)
   (distance(r.pos, o.pos)<= 20)
   (rel_angle(r.pos, o.pos, r.heading) = 90)
   (r.speed = r.speed_max))
  ;; behaviour preconditions
  ((duration 10)
   (constant(r.heading))
   (constant(r.speed)))
  ;; effects
  ((has_film ?r ?o))
)
```

The significance of the attitude and behaviour preconditions is : the rover must position itself at a distance ranging between 10 and 20 meters from the objective, and must be perpendicular to the objective. During the action, the rover has to maintain its course and its speed.

As these contraints do not describe, in all the cases, a simple point of the environment, a failure of the attitude preconditions satisfaction does not significate that the action is impossible. Figure 7 presents the tests performed by the planning system for checking the operator preconditions. During the planning process, the main planner sends the geometrical constraints described in the attitude preconditions through a request to the path planner in order to know if the rover can be correctly positioned. If the answer is positive and if the action requires some movement - or more globally a specific behaviour - during its achievement, the task planner sends a new request to the path planner containing the related geometrical constraints. If all these preconditions are satisfied then the planner can apply the operator effects to the current state of the world. Otherwise the task planner backtracks and tries to apply another action. If the behaviour preconditions are unsatisfiable, the path planner search another solution satisfying the attitude preconditions. Indeed, a solution for the behaviour preconditions depends on the solution found for the attitude preconditions.

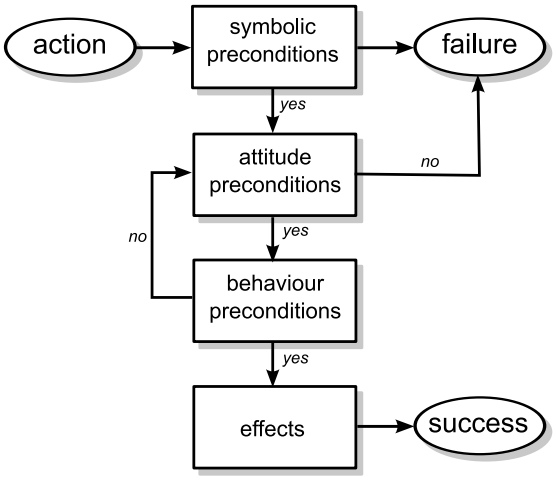


Figure 7: Decomposition of an operator precondition test

3.3 Planning algorithm

The HTN planning algorithm implemented in the architecture is represented on the algorithm 3.1.

The planning problem can be represented as a tree called a planning tree. A tree node contains the current state of the world and the list of the remaining tasks to achieve. Thus at the initialization stage, the planning tree is composed of a unique node representing the initial state and the high-level tasks to plan. The objective of the algorithm is to develop the tree until obtaining a leaf node, ie. a node whose task list is empty. The development of a node into a subnode is done by applying a planning operator or a method.

Initially, the algorithm tests if the task list is empty (line 2). In this case, the current node is a leaf node and a solution is found. Otherwise, the algorithm tries to achieve the first task.

If the task t_0 is primitive, ie. non decomposable (line 4), then for each operator o of the set of operators O, the algorithm tries to unify the head of the operator with the task in order to link the free operator parameters with the instancied task parameters. Then the operator preconditions are tested with the current state (line 8). If the unification is possible, it computes the possible substitutions σ which is the result of the unification. At this stage, the attitude and behaviour preconditions must be satisfied. a requests for the attitude precondition is sent to the path planner. If the specialized reasoner sends back a positive answer, another request is sent in order to try to satisfy the behaviour preconditions. Then the procedure applies the substitued effects to the current state of the world (line 14) and creates a new node containing this resulting state and the remaining tasks. Finally the algorithm is called recursively on the new node (line 15).

If the task t_0 is compound (line 26), the algorithm tries to apply a method m contained in the set of methods $\mathcal{M}$ by testing the preconditions of each decomposition d (line 31) of the choosen method m (line 29). For each substitution σ, a new node is created. This node contains the current list of tasks in which t_0 is replaced by the reduction of d (line 35), ie. the task list resulting of the method decomposition. Then the procedure is called recursively on the new node (line 36).

3.4 Interaction between components

From the attitude and behaviour preconditions, the specialized planner must find a path allowing the positioning of the robot for the achievement of the action and then move it during the achievement itself. If there is no solution, *i.e.* no possible position or movement, it sends back a failure message. In this case, the task planner must find another action to plan.

This exchange between the planners is done through communication primitives, ie.with the use of a common vocabulary. This language allows to identify *concepts* (eg. agent, object), *global properties* (eg. duration), *concept properties* (eg. r.heading, o.position) and *rules* (eg. =, constant) contained in the requests.

Concepts allow to identify the agents and physical objects of the environment, which will be handled by the planners. With each concept we associate a set of properties called the *concept properties*. These properties indicate the agent and object properties handled. *Global properties* specify properties of a movement subproblem as, for instance, the maximal duration of a robot move. *Rules* allow to define the geometric constraints of the agents compared to the objects of the environment, as well as the kinematic constraints of the robot. They are applied to the concepts properties and global properties.

The *concept properties* are specific to each agent. But it is possible to define *ontologies of properties*. We may have one ontologie for each type of mobile robot. A specific agent is able to interpret only a fixed number of properties concerning its own abilities. For instance, a planetary rover will not be able to interpret the *vertical speed property*. However, this concept property may be essential for an unmanned aerial vehicle. The designer is in charge of defining a planning domain containing only properties interpretable by the specialized planner.

Algorithm 3.1: DevelopNode(n, O, $\mathcal{M}$)

```
1  Soit n = (s, < t_0, ..., t_n >) ;
2  if < t_0, ..., t_n > == nil then
3      return n ;
4  else if t_0 is primitive then
5      foreach operator o ∈ O do
6          θ ← Unify(name(t_0), name(o)) ;
7          if θ ≠ null then
8              Σ ← FindSubstitution(preconditions(o), s, θ) ;
9              foreach substitution σ in Σ do
10                 ac ← Request(attitudeCond(o, σ), PathPlanner) ;
11                 if ac == true then
12                     bc ← Request(behaviourCond(o, σ), PathPlanner) ;
13                     if bc == true then
14                         s' ← apply the effects of o to s ;
15                         n' = DevelopNode((s', < t_1, ..., t_n >), O, M) ;
16                         if n' ≠ null then
17                             add n' as subnode of n ;
18                             return n ;
19                         end
20                     end
21                 end
22             end
23         end
24     end
25     return null;
26 else if t_0 is compound then
27     foreach method m ∈ M do
28         θ ← Unify(name(t_0), name(m)) ;
29         if θ ≠ null then
30             foreach decomposition d ∈ decompositions(m) do
31                 Σ ← FindSubstitution(precond(d), s, θ) ;
32                 if Σ ≠ null then
33                     foreach substitution σ in Σ do
34                         s' ← apply the effects of d to s ;
35                         < t_{n+1}, ..., t_{n+i} > = reduction(d) ;
36                         n' = DevelopNode((s', < t_{n+1}, ..., t_{n+i} > + < t_1, ..., t_n >), O, M) ;
37                         if n' ≠ null then
38                             add n' as subnode of n ;
39                             return n ;
40                         end
41                     end
42                 end
43             end
44         end
45     end
46     return null;
47 end
```

These different elements of the communication language must be encapsulated in messages which will be exchanged between the planners. The messages are expressed in the form of requests and advices.

3.4.1 Requests and queries

We can distinguish two types of requests: *planning requests* and *system requests*. Planning requests allow the general purpose planner to ask the specialized planner to find movements satisfying the given geometric constraints.

Definition 3.1 (Planning request)
A planning request R is defined as a tuple $< Type(R), Agent(R), Id_{action}(R), \mathcal{C}(\mathcal{R}) >$

- $Type(R)$ *is the type of the request. For example, attitude to define an attitude precondition request, or behaviour for a behaviour precondition request ;*
- $Agent(R)$ *allows to know which agent is concerned by the request in the case of a multi-agent problem ;*
- $Id_{action}(R)$ *is an action Id allowing to maintain a coherence between the task planner and the path planner resolution ;*
- $\mathcal{C}(\mathcal{R})$ *is the set of geometric constraints to be activated before or respected during the robot movements.*

An example of planning request is :
`request(attitude, robot1, #1, {distance(r.pos, o.pos)>=10),...})`

System requests is used by the interface in order to give some instructions to the different planners. These instructions are used to initialize, to guide and to finalize the plan construction.

Queries are messages sent by the task planner to the path planner in order to ask for an *advice*. They are used when the task planner is in front of a choice, as for example, *do something at point A or do something at point B*. The answer of the specialized planner is called an *heuristic advice*.

For example, in order to choose between doing an action at point A or at point B, according to the current robot position, the query sent by the task planner will be :
`query(min_distance, robot1, {A,B})`

3.4.2 Advices

Advices are messages sent by the path planner to the task planner. We can distinguish three types of advices : *heuristic advices* which are answers to the high-level planner queries, *optimization advices* which expressed some proposals of the specialized planner in order to optimize the final plan, and *repair advices* which are used to propose movement alternatives allowing to avoid a deep backtrack of the task planner.

The heuristic and optimization advices can be seen as optional requests. If they are not taken into account, the planners can still construct a valid plan but this one will be less optimized.

Definition 3.2 (Advice)
An advice A is defined as a tuple $< Type(A), Content(A), Agent(A), set\{Id_{action}(A)\} >$

- $Type(A)$ *is the type of the advice : heuristic, optimization, repair ;*

- *Content*(A) *is the content of the advice ;*
- *Agent*(A) *allows to know which agent is concerned by the advice in the case of a multi-agent problem ;*
- *set*$\{Id_{action}(A)\}$ *is a set of Ids for actions concerned by this advice.*

An example of *optimization advice* for the task planner is:
`advice(optimization, reverse, robot1,{#2,#3})`

In this example, the path planner asks the general purpose planner if it can reverse the realization order of task #1 and task #2 (because task #1 and #3 must be realized on the same objective).

3.5 Example of interactions between the planners

A robot must take pictures of objectives A (action #1) and B (action #2), and sample rock at point A (action #3). This example is represented on the figure 8.

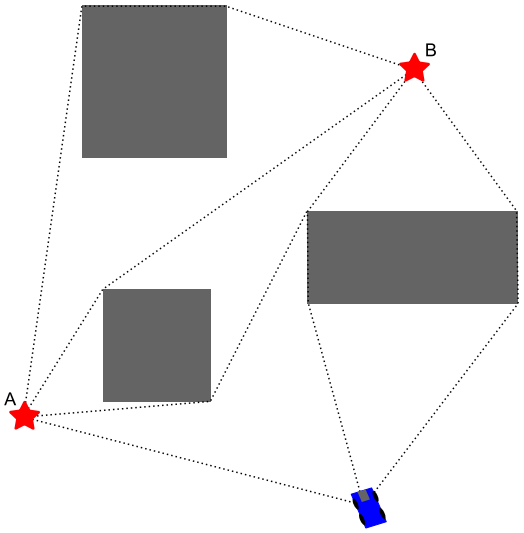


Figure 8: The example used

The sequence diagram (figure 9) represents the sequence of exchanged messages between the different modules during the resolution of the example problem.

After the initialization phase (1), the task planner tries to plan the first action (*film the objective A*). It sends a request (2) to the path planner in order to test the attitude preconditions of this action. This request contains the set of constraints expressed in the planning operator `film_objective`. The path planner replies with a positive answer (4), *i.e.,* required movements are possible in order to position the robot for starting the action achievement. So, the task planner sends a second request which aims at verifying the behaviour preconditions (6).

In the same way, the two planners exchange a set of messages in order to plan the action `film_objective` at the point B (10-17), and the action `sample_rock` at the point A (18-25).

At this point, the task planner sends a end-message to the interface (34), ie. there is no more task to achieve. The general purpose planner has successfully decomposed its high-level tasks into primitive actions which can be achieved by the robot. The interface informs the specialized path planner of the completion of the mission planning process.

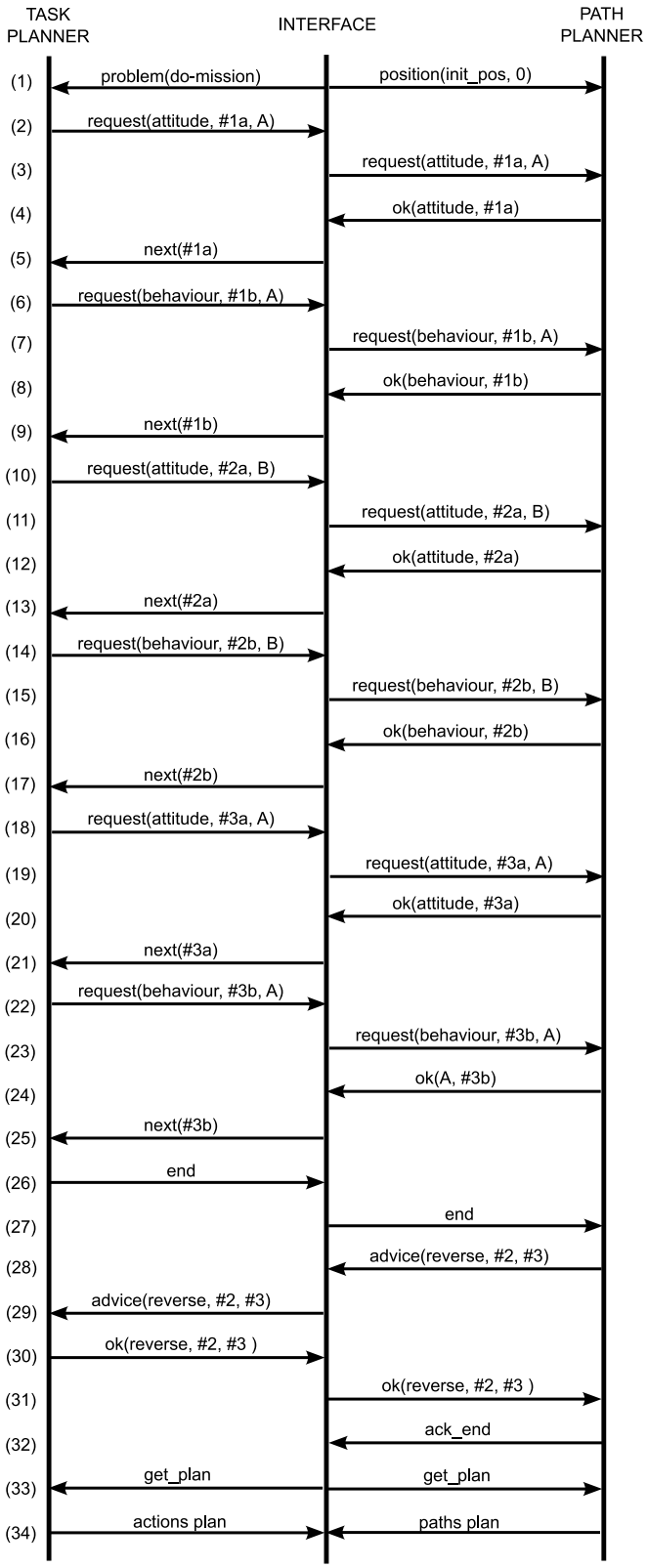


Figure 9: Sequence diagram of the exchanged messages during the construction of the plan (using a simplified formalism).

As the specialized planner has calculated all the needed movements, it can now try to optimize the entire robot trajectory. Indeed, the robot visits the point A twice. This is due to the local search of movements without having a global vue of the planned trajectory. So, the path planner proposes an optimization advice to the task planner (28). This advice proposal is to reverse the achievement order of actions #2 and #3. Indeed, since the point A was already visited, the inversion of these two tasks will allow to optimize the solution in term of traveled distance. The sequence of the three actions is expressed as an unordered sequence, *i.e.,* these actions can be planned in any order. The task planner reply with a positive answer to this advice. Indeed, changing the order of achievement of these actions does not impact the rest of the symbolic plan (30).

The final plan can be constructed and sent back to the mission management system (33). This construction is done by assembling the plan of actions produced by the symbolic reasoner and the plan of movements produced by the geometric reasoner. The correspondence between symbolic actions and movements is done with the use of action Id which are present in the exchanged requests.

Compared to a traditional hierarchical planning architecture in which a symbolic plan is found before trying to satisfy the entire set of geometric constraints, the proposed architecture is able to react immediatly to a failure of some geometric preconditions.

4 CONCLUSION

Current planning techniques in robotic applications present open issues. One of these issues is how to efficently bind a symbolic reasoning for the selection of actions the robot has to achieve and a geometric reasoning for the computation of the robot movements. In this article, we tried to propose a solution to this problem.

We have made a comparison between the classical planning approach in which a task planner tries to solve entirely the problem and a hybrid model in which the path finding problems are delegated to a specialized path planner. The results show a significant improvement of the computation time when a path planner is used. Even if the results seem to be obvious, they show at which point the use of a specialized planner for treating subproblems improves the computing time and the quality of the solution. Then we have presented different methods to bind a symbolic planner with a specialized reasoner. We have shown that an interleaved approach in more efficient in term of computation time and messages exchanged between planners than the hierarchical method, even if the backtrack process is well informed, ie. done at the level of the task that is geometrically impossible.

From these observations, we have proposed a planning architecture allowing to deal with robotics problems in which a mobile robot has to perform some tasks in the environnement, and illustrated it through an example of data acquisition mission for a mobile robot. This architecture is based on a strong coupling between a symbolic and a geometric reasoning. The symbolic reasoning is made by a hierarchical task planner allowing to express the tasks to perform as recipes (through the use of methods). The geometric aspects of the problem are handled by a specialized path planner.

The functioning of our architecture is as follow : the task planner calls the path planner by sending requests expressing the necessary conditions for the agent to move in the environment according to the constraints. These constraints are formulated using specific preconditions of the planning operators : the attitude preconditions and the behaviour preconditions. The communication between the two planners is done through an interface with the use of a common vocabulary. The exchanged messages are expressed in the form of requests containing the necessary geometric constraints to be respected in order to achieve the actions. The other type of exchange concerns the sending of advices allowing, on the one hand, to guide the symbolic plan construction based on information given by the path planner and, on the other hand, to re-schedule the

already-planned actions in order to optimize the mission.

Thereafter, we propose to extend our architecture in a multi-agent context in which several mobile robots will have to cooperate to accomplish the mission. However, since the notion of optimization for a multi-robot mission plan is more complex than for a single robot, we will have to study the efficientness of our architecture in this case. We will especially study over-subscribed problems in which the planning architecture will have to choose objectives in a huge set of possible actions. We will also study the architecture's ability to react in the case of a necessary replanning due to events occuring during the execution as well as how the already computed plans can be re-used to efficiently repair the current failure. Finally we will implement our propositions within the framework of search-and-rescue missions for a team of unmanned aerial vehicles.

References

[1] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

[2] B. L. Brummit and A. Stenz. GRAMMPS: A generalized mission planner for multiple mobile robots in unstructured environments. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1564–1571, May 1998.

[3] S. Cambon, F. Gravot, and R. Alami. aSyMov: Towards more realistic robot plans. In *International Conference on Automated Planning and Scheduling*, 2004.

[4] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[5] K. Erol, J. Hendler, and D. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, pages 249–254, 1994.

[6] M. Fox and D. Long. PDDL 2.1 : an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[7] M. Ghallab. An overview of planning technology in robotics. In *27th Annual German Conference on Artificial Intelligence*, 2004.

[8] M. Ghallab, D. Nau, and P. Traverso. *Automated planning : Theory and Practice*. Morgan Kaufmann, 2004.

[9] A. Howe and E. Dahlman. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research*, 17:1–33, 2002.

[10] S. Kambhampati, M. Cutkosky, J. Tenenbaum, and S. Lee. Integrating general purpose planners and specialized reasoners : case study of a hybrid planning architecture. In *IEEE transactions on Systems, Man and Cybernetics*, pages 1503–1518, 1993.

[11] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration for fast path planning. In *IEEE Int. Conf. on Robotics and Automation*, volume 3, pages 2138–2145, May 1994.

[12] B. Lamare and M. Ghallab. Integrating a temporal planner with a path planner for a mobile robot. In *AIPS Workshop Integrating planning, scheduling and execution*, pages 144–151, 1998.

[13] J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[14] S. LaValle and J. Kuffner. Randomized kinodynamic planning. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 473–479, Oct 1999.

[15] D. Nau. Current trends in automated planning. *AI Magazine*, 4(4):43–58, 2007.

[16] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. SHOP2 : An HTN planning system. *Artificial Intelligence Research*, 20:380–404, 2003.

[17] D. Nau, H. Munoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *the International Joint Conference on Artificial Intelligence (IJCAI)*, Aug. 2001.

[18] G. Rabideau, T. Estlin, S. Chien, and A. Barrett. A comparison of coordinated planning methods for cooperating rovers. In *AIAA Space Technology Conference and Exposition*, pages 133–140, Sept. 1999.

[19] G. Ramkumar. An algorithm to compute the minkowski sum outer-face of two simple polygons. In *twelfth annual symposium on Computational geometry*, pages 234–241, 1996.

[20] D. Smith. Choosing objectives in over-subscription planning. In *14th International Conference on Automated Planning and Scheduling*, jun 2004.

[21] Third International Planning Competition. Hosted at the Artificial Intelligence Planning and Scheduling (AIPS) conference. http://planning.cis.strath.ac.uk/ competition/, 2002.

[22] D. Wilkins and M. desJardins. A call for knowledge-based planning. *AI Magazine*, 22(1):99–115, 2001.

[23] F. Zacharias, C. Borst, and G. Hirzinger. Bridging the gap between task planning and path planning. In *IEEE International Conference on Intelligent Robot ans Systems*, pages 4490–4495, Beijing, China, October 2006.