A Hybrid Approach to Intricate Motion, Manipulation and Task Planning

Stéphane Cambon, Rachid Alami, Fabien Gravot

A Hybrid Approach to Intricate Motion, Manipulation and Task Planning

Stéphane Cambon SCAMBON@GMAIL.COM

Rachid Alami RACHID.ALAMI@LAAS.FR
CNRS; LAAS; 7, avenue du colonel Roche
Université de Toulouse ; F-31077 Toulouse, France

Fabien Gravot FGRAVOT@LAAS.FR
CNRS; LAAS; 7, avenue du colonel Roche
Université de Toulouse ; F-31077 Toulouse, France

Abstract

We propose a representation and a planning algorithm able to deal with problems integrating task planning as well as motion and manipulation planning knowledge involving several robots and objects. Robot plans often include actions where the robot has to place itself in some position in order to perform some other action or to "modify" the configuration of its environment by displacing objects. Our approach aims at establishing a bridge between task planning and manipulation planning that allows a rigorous treatment of geometric preconditions and effects of robot actions in realistic environments. We show how links can be established between a symbolic description and its geometric counterpart and how they can be used in an integrated planning process that is able to deal with intricate symbolic and geometric constraints. Finally, we describe the main features of an implemented planner and discuss several examples of its use.

Keywords: planning formalism, task planning, motion planning, manipulation planning.

1. Introduction

This paper aims at enriching the expressive power and the problem-resolution capabilities of symbolic task planners to allow their effective use for realistic robotics problems.

While symbolic task planners have been drastically improved to solve more and more complex symbolic problems (Hoffmann et al. (2006), the difficulty of successfully applying such planners to robotics problems still remains. In such planners, actions such as "navigate" or "grasp" use abstract notions of the reachability of goal configurations that might result in finding plans that cannot be executed by the robots. This is due to the gap between the representation they are based on and the physical environment (see Lozano-Perez et al. (1987)).

On the other hand, practical motion planning techniques (LaValle (2006)), and more precisely manipulation planners (Alami et al. (1990); Siméon et al. (2003)), have experienced very important

progress. The goal of the work described in this paper is to devise a task planner that is aware of the geometrical and topological constraints and consequences of its actions in the geometrical environment. This feature might be crucial in robotics when we wish robots to be able to manipulate objects and/or to reach positions where specific actions can be performed in a relatively constrained setting.

We propose to enrich the capabilities of task planners with those of manipulation planners, to investigate the link between the two representations and to establish an architecture where the search for a solution interestingly performs a parallel and coordinated search in the two schemes.

Real world robots never operate in environments with complete and exact knowledge of the system state and perfect execution of planned actions(Alami et al. (1998)). Consequently, our motivation here is of fundamental nature. Even though we place ourselves in an ideal situation where we assume complete knowledge of the environment state with no consideration of uncertainty issues, we think that still there are useful insights to obtain through a process in which we try to provide a systematic way to link geometric and symbolic constraints. To make things concrete, we would like to come up with a planner that not only produces a set of partially ordered high-level actions but also a full instantiation that indicates the locations of robots and objects as well as the paths for the robots and their synchronization.

1.1 The radio-switch example

In order to illustrate our approach, we will use a class of problems, that we have called *radio-switch* as an example all over the paper. Figure 1 illustrates one problem belonging to this class. This problem is intentionally very simple.

It involves two robots: $forklift$ and $armbot$ (a mobile robot with an arm) and one object: $cbox$ (for communication-box). $cbox$ has a wireless modem. The mission is defined by the following goal state: data has been received and $cbox$ is placed in the other room. In order to receive the data, $cbox$ has to be placed in a specific area. The data are sent continuously after the switch has been pushed. Only $armbot$ is able to perform this push operation and only $forklift$ can manipulate $cbox$.

Variants of the radio-switch problem class are created by giving an initial and a final collision-free configuration to each robot and object. Note that it is not necessary nor desirable to provide a complete specification of the final state. Depending of these configurations, the valid plans may be drastically different. For example, it might be necessary for $forklift$ to perform re-grasping operations in order to satisfy the $cbox$ goal position and orientation (represented by an arrow on figure 1). For some other initial states, as shown in figure 1, $cbox$ might block the way to the switch: in such a case a solution plan should be found that imposes to displace the object to a suitable intermediate configuration.

1.2 Related works

Implementation decision at the so called “task-level” - selecting and scheduling robots actions - and at the “geometric level” - motion, grasp configuration - has been identified and is still a challenge in robotics.

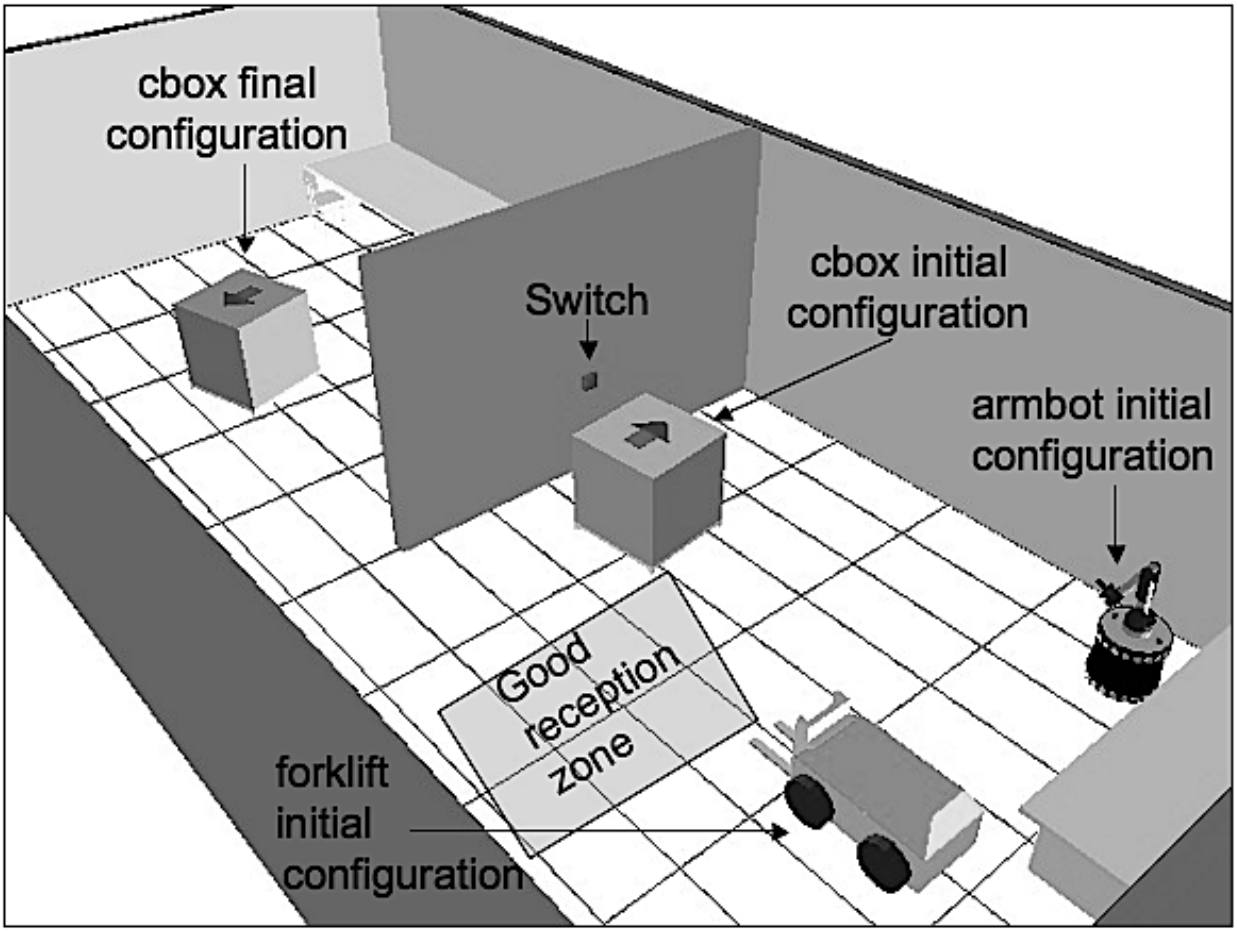

Figure 1: One radio-switch problem.

Experience with high-level planners in robotics have been tried and have given interesting results and valuables insights with respect to architectural issues as well as links between deliberation and on-line reactive planning. However this has been done only in contexts and situations where there is no intricate dependencies between the “symbolic” considerations and the “geometric” ones, leaving place to the use of independent resolution schemes that can run at different levels of abstraction (Fink and Veloso (1996); Alami et al. (1998)).

Indeed, the task planners, based on a discrete sets of actions, cannot handle the complexity of a realistic geometric world. This is due to the difficulty of representing by symbols the infinite number (or, at least, the very large number of discretized choices) of the robots motions in a continuous world or the infinite number of the objects intermediate configurations. A plan generated by the task planner may turn out to be infeasible when developed at the motion planning level (Latombe (1991)).

Several very interesting contributions consisted in systems that formally integrate high-level planning with continuous control primitives (Rugg-Gunn and Cameron (1994); Fainekos et al. (2005)).

However, these approaches do not involve an interleaving between the two levels at the planning phase.

There are also a number of interesting studies that have focused on planning for assembly. The set of grasps, and more generally of objects configurations, are represented by constraints in Halperin et al. (1998) or by a graph in Hutchinson and Kak (1990); Sanderson et al. (1990); Cao and Sanderson (1994); Tung and Kak (1996). In such problems, the elementary arcs that constitute the graph (i.e. the instantiated basic actions) are given to the problem solver.

On the other hand, motion planners do not deal with abstract notions. A key issue on which our approach is built is the so-called manipulation planning formalisms and algorithms. This allows one to draw a strong link between geometry and tasks. For instance, depending on how an object is grasped, it can be put or not be put on a table. It is not easy to clearly separate the task "put the object on the table" from the grasp configuration to choose. Several authors have discussed this subject (Lozano-Perez et al. (1987), Laugier and Troccaz (1985), Mazon et al. (1990)). These systems have been created for assembly problems in which the number of configurations and grasps per object is generally small.

In our case we wish to deal with mobile manipulators problems and keep a continuous representation of the environment geometry. We do not require to identify or pre-compute all the constraints before planning: this identification will be done incrementally during the planning process itself.

Several methods and algorithms have been proposed to address problems that involve discrete (qualitative) changes and continuous motion primitives (Wilfong (1988); Alami et al. (1990); Chen and Hwang (1991); Koga and Latombe (1994); Ahuactzin et al. (1998); Nielsen and Kavraki (2000); Lingelbach (2004); Bretl (2006); Plaku et al. (2007); Saut et al. (2007)). However, all these methods address only motion or manipulation actions and conditions to link, or to switch between motion commands.

Our concern here is to address in a coherent manner all types of actions, from actions that have no geometric constraints to actions that have geometric and symbolic constraints and effects. The ambition is a scheme to help tackle and solve problems that combine complexity at the geometric level and complexity at the symbolic level (several types of actions, several objects, several robots). Such a scheme should allow, for instance, to provide reasoning mechanisms that can guide the overall search process so as to avoid or reduce combinatorial motion planning sub-problems.

1.3 Overview

In order to reason on geometric preconditions and consequences of actions, we rely on properties expressed in the overall Configuration Space of the system. This space lets us express the geometric constraints for mobile robots and movable objects in a 2D or a 3D[1] world. The main idea here is to relate in a principled manner the symbols representing places or relations of the world with sub-manifolds of the Configuration Space. With this idea, we will be able to express formally the preconditions needed to apply an action and the effects of this action not only at a symbolic level but also with regard to the geometry of the environment where it will be applied. Thus, we will describe a search space which can be explored by a task planner to synthesize a plan. One of

1. or more, Configuration Spaces definition are not limited to a given number of dimension of the workspace

our ambitions in this paper is to link, when necessary, the task planning and the motion planning definitions.

In section 2, we give a formal representation to express task planning problems that integrate geometric and kinematic constraints. The model links symbols and relations to sets of configurations. The approach is generic. We do not make any hypothesis on how the configuration space can be partitioned in sets of configurations. This is the reason why, in the third section, we identify specific subsets of the Configuration Space which play a role in the case of a multi-robot manipulation problem. These results will help us to more precisely understand how to define the problem introduced in section 2 as an instance of a manipulation problem involving several robots. This is the topic of section 3. The two following sections deal with the resolution of such problem. In section 4, we first introduce some adaptations of the Probabilistic Roadmaps Methods. The main point is the use of several roadmaps and a two-step algorithm to break the complexity brought by highly dimensional Configuration Spaces. In section 5, we present the task planner that we have developed to solve the class of problems introduced. The core algorithms are detailed and an empirical evaluation of the overall system is presented.

2. Extension of Task Planning definitions

2.1 Formal problem statement

Besides the symbolic pre-conditions and effects of task-planning operators, we would like to also take into account the geometric constraints and effects of actions in a realistic 3D environment. We extend the definitions of the task planning state space. To do so, we rely on the definitions used in Ghallab et al. (2004) and on the Configuration Space definitions (Lozano-Pérez (1983)) that have been extended in Alami et al. (1990); Siméon et al. (2003) to account for manipulation problems.

In order to have a self contained set of definitions, the classical definitions are introduced when needed. Note that we present here our definitions as an extension of the state space task planning domains and problems, but it would also be possible to present them as an extension of the motion planning problem.

We begin by introducing the definition of a symbolic planning domain.

Definition 1: the symbolic planning domain

- *A **symbolic planning domain** is a state transition system Σ_{sym} on a propositional logic language $\mathcal{L}$. We have $\Sigma_{sym} = (S_{sym}, A, \gamma_{sym})$.*
- A **symbolic state** $sym_s \in S_{sym}$ is a conjunction of (positive) propositions. We assume here with the Close World Assumption.
- *An **action** $a \in A$, is a triplet $\{name(a), precond(a), effects(a)\}$ where*[2]

2. We adopt here a set-theoretic representation (Ghallab et al. (2004)) for task planning. Each state of the world is a set of propositions, and each action is a syntactic expression specifying which propositions belong to the state in order for the action to be applicable, and which propositions the action will add or remove in order to make a new state of the world.

  - $name(a)$ *is the symbol used for this action.*
  - $precond(a)$ *is a conjunction of positive propositions* $precond^+(a)$ *and negative propositions* $precond^-(a)$*.*
  - $effects(a)$ *is a conjunction of positive propositions* $effects^+(a)$ *and negative propositions* $effects^-(a)$. that should be added or deleted after the action has been applied.

- *The* ***transition function*** γ_{sym} *is built as follows: An action* $a \in A$ *can be applied on a symbolic state* $sym_s \in S_{sym}$ *if and only if:*

  (1-1) $precond^+(a) \subset sym_s$

  (1-2) $precond^-(a) \cap sym_s = \emptyset$

  The application of an action a *on the state* sym_s *results in a new state* $sym_{s'}$ *through the* γ_{sym} *function. We have* $\gamma_{sym}(s, a) = s'$ *with:*

  (1-3) $sym_{s'} = (sym_s - effects^-(a)) \cup effect^+(a)$

It is this definition that we want to extend in order to integrate geometric constraints and consequences of motions and manipulations in a representation of the geometry. Let us suppose that, besides the state transition system Σ_{sym} we have a two-dimensional or three-dimensional geometric and kinematic representation of the world: a set of robots, of "movable" objects and of obstacles which define the system Configuration Space CS.

Definition 2: the Configuration Space

- *In a two-dimensional or three-dimensional environment involving a static geometric environment and several movable mechanical systems, rigid or not, a* ***configuration*** *completely specifies the position and orientation of all the components of the composite mechanical system.*

- *We call* ***composite mechanical system*** *the geometric movable object that groups all the distinct geometric movable objects (in our case, the set of robots and objects that can be moved by robots).*

- *The composite* ***Configuration Space*** CS *is the set of all these configurations.*

For instance, if our geometric world is three dimensional and if it involves a 9 dofs (i.e. degrees of freedom) robot and a 6 dofs free-flying object, the dimension of the composite mechanical system is 15.

Definition 3: the Free Configuration Space

The ***Free Configuration Space*** CS_{free} *the subspace of* CS *which contains all the configurations of the composite mechanical system which are not in collision with the static geometric environment.*

Now we define the motions of the composite mechanical system. We introduce a reachability function.

Definition 4: the reachability function

*We define the **reachability function** f_{reach} for the composite mechanical system. This function defines the possible motions without collision of the mechanical system. A motion in the geometric world corresponds to a **path** in CS. For each $c \in CS_{free}$ we have $f_{reach}(c) = cs_c^{reach}$ where cs_c^{reach} is the subspace of CS_{free} grouping all the configurations that are reachable from c by a motion without any collision along the path.*

Now we want to construct a state space where the actions involving motion or manipulation, and the potential paths in CS associated to them, have a geometric reality. The preconditions and effects of such actions will take into account the reachability function. We link CS with Σ_{sym} through propositions of $\mathcal{L}$ that will be associated to subsets of CS. To identify in $\mathcal{L}$ these symbols we introduce a specific set of proposition: $PLACE$.

Definition 5: the $PLACE$ set

Let the set $PLACE$ be a subset of the set of the propositions of the world. Each proposition of this set is named a $place$ proposition. Each proposition p that belongs $PLACE$ is linked through a relation $\mathcal{R}$ to a subset of CS.

Now we formally define the $\mathcal{R}$ Relation.

Definition 6: the $\mathcal{R}$ Relation

***The relation** $\mathcal{R}$ is a relation between propositions p of the set $PLACE$ and a subset of configurations $cs_p \subset CS$.*

Let us take an example of a $\mathcal{R}$ relation with one element. Let the world be a 3D static geometric world, and let an object named *leg* be a free-flying (6 dofs) movable object. The geometric object *leg* describes a 6-dimension Configuration Space CS. Now, let $\mathcal{L}$ be a propositional logic language containing some $place$ propositions. Let cs_{on_table} be a set of configurations of CS as defined in figure 2. We introduce a relation $R = \{$*leg_on_table*$, cs_{leg\_on\_table}\}$. The relation gives us a geometric meaning of the proposition *leg_on_table*.

Now we introduce the definition of a state. For us a state has a symbolic and a geometric part.

Definition 7: the state

A **state** s is a couple (sym_s, cs_s) where sym_s is a symbolic state as in definition 1, and $cs_s \in CS$ is its geometric "counterpart".

Now we introduce two definitions that we will use in the final planning domain definition. The first definition concerns the *set of configurations associated to a state* s, cs_s^{def}. This set defines in what set of CS cs_s has to be. The second definition introduces the *set of reachable configurations* cs_s^{reach} of a state s.

Definition 8: the set of configurations associated to a state

If a symbolic part sym_s of a state $s = (sym_s, cs_s)$ contains n propositions p_i $(0 \leq i < n)$ such that p_i belongs to the set $PLACE$ then the **set of configurations associated** to s is:

$$cs_s^{def} = (\bigcap_{i=0}^{n} cs_{p_i} / (p_i, cs_{p_i}) \in R)$$

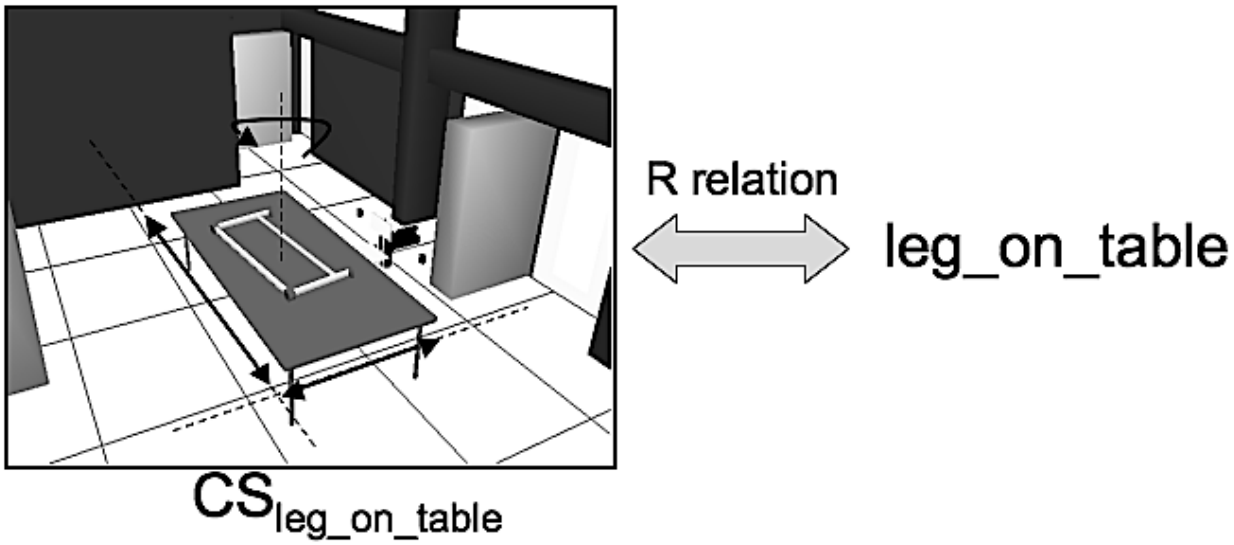

Figure 2: An example of $\mathcal{R}$ relation. Here $cs_{leg\_on\_table}$ is a three-dimensional sub-manifold including the configurations (positions and orientations) that correspond to a stable placement of the leg on the table.

If the symbolic part sym_s of a state $s = (sym_s, cs_s)$ does not contain any proposition p that belongs to the set $PLACE$ then: $cs_s^{def} = CS$, *which is the neutral element of the intersection in* CS.

In the same example of the *leg* and the table, if we have a state $s = (sym_s, cs_s)$ such that sym_s = *(leg_on_table)* then the set of configurations cs_s^{def} associated to s is $cs_{leg\_on\_table}$. This set is clearly decomposable in two subsets corresponding to the leg lying on one face or on the other. We will see later how this connected components will be detected and treated. Now let us imagine that we have two objects *leg1* and *leg2* and a state $s = (sym_s, cs_s)$ such that sym_s = *(leg1_on_table)* $\wedge$ *(leg2_on_table)* and with *(leg1_on_table)* and *(leg2_on_table)* belonging to the set $PLACE$, then the set of configurations cs_s^{def} associated to s is $cs_{(leg1\_on\_table)} \cap cs_{(leg1\_on\_table)}$. In other words, it is the set of configurations where the two objects are on the table. Some of these configurations (corresponding to collisions between the two legs) do not belong to CS_{free}. Note that if it does not exist at least one configuration of cs_s^{def} which belong to CS_{free}, the state s will be unreachable.

Definition 9: the set of reachable configurations of a state s

For a given state $s = (sym_s, cs_s)$ we define the **set of reachable configurations** cs_s^{reach} as the set of all the configurations that are reachable (i.e. through their reachability function) from any configuration of cs_s.

We have now all the definitions needed to formulate a planning domain that is aware of the environment geometry.

Definition 10: The planning domain

- A **planning domain** is a state transition system Σ on the propositional logic language $\mathcal{L}$, a configuration space CS (induced by a geometrical and kinematic environment description) and a relation $\mathcal{R}$. We have $\Sigma = (S, A, \gamma)$.
- A **state** $s \in S$ is a couple (sym_s, cs_s). sym_s is a conjunction of propositions. cs_s is a subset of CS.
- An **action** $a \in A$, is a triplet $\{name(a), precond(a), effect(a)$ as defined in definition 1.
- The **transition function** γ is built as follows:

  An action $a \in A$ can be applied on a state $s \in S$ if and only if:

  (10-1) $precond^+(a) \subset sym_s$

  (10-2) $precond^-(a) \cap sym_s = \emptyset$

  (10-3) $cs_s \neq \emptyset$ (s is valid)

  The application of an action $a \in A$ on the state s results in the state s' through the γ function. We have $\gamma(s, a) = s'$ with:

  (10-4) $sym_{s'} = (sym_s - effects^-(a)) \cup effect^+(a)$

  (10-5) $cs_{s'} = cs_s^{reach} \cap cs_{s'}^{def}$

One can observe that (10-1), (10-2) and (10-4) correspond to the classical definition of the task planning (definition 1). (10-3) has been added to ensure that the state has a "geometric existence". (10-5) causes the creation of the subspace attached to s'. This new subspace must belong to $CS_{s'}^{def}$ and must be reachable from the state s.

Definition 11: a planning problem

A **planning problem** is a triplet $P = (\Sigma, s0, g)$ where:

- $s0$ is the initial state.
- g is a conjunction of propositions.
- the goal states are specified as
  $S_g = \{s \in S / sym_s \; satisfies \; g \; and \; cs_s \neq \emptyset\}$

Generally in our problems, we will choose the initial state $s0 = (sym_{s0}, cs_{s0})$ such that $cs_{s0} = \{c\}$, a unique configuration.

Note that, following our definition, if $s = (sym_s, cs_s)$ is a state obtained by n applications of actions from $s_0 = (sym_{s_0}, cs_{s_0})$ then

$$cs_s \subseteq CS_{free}$$

This means that successive application of actions maintain the geometric system in the free space.

Important remark. Note that, with our definitions, we also have a more general formalization for planning. We can easily express:

- A classical motion planning problem. In such case, $S = \{s0, s1\}$, $A = \{a\}$, $\gamma = (s0, a, s1)$.
- A classical task planning problem. In such case, $PLACE = \emptyset$.

2.2 Practical issue

In realistic problems, the construction of the exact topology of CS is out of reach. One possibility (our choice) is to search for a plan without building an explicit representation of CS. Symbolic plans that respect (10-1), (10-2) and (10-4) are first built. These symbolic plans are solutions to a relaxed problem where the geometric preconditions and effects are ignored. Then a solution is obtained if we find at least one configuration belonging to cs_s for each state $s = (sym_s, cs_s)$ traversed by one of these symbolic plans. The search will balance between the construction of new symbolic plans and the search for configurations to validate the states.

The next section will give definitions on the topology of the CS where actions include motion and manipulation involving several robots and several objects.

We will see how a $\mathcal{R}$ relation can be defined to comply with the manipulation context. Building an adequate $\mathcal{R}$ relation is indeed the other big practical issue.

Moreover, these definitions will help us propose techniques to break the complexity induced by the high dimensionality of CS by exploring separately the configuration spaces of each robot or movable object.

3. Topology of the Configuration Space in the Multi-Robot Manipulation Problem and consequences on the definition of problems

3.1 The Multi-Robot Manipulation Problem

The multi-robot manipulation planning problem involves robots and objects among fixed obstacles. The objects cannot move by themselves; either they are carried by robots or they stay in some stable placement. It has been shown that the manipulation planning problem is a constrained instance of the coordinated motion planning problem. We follow and extend the definitions presented by Siméon et al. (2003) to the case of several robots and several objects.

Formally, in our three-dimensional workspace, we have r robots R_i which have n_i degree of freedom respectively. We have also m movable rigid objects which have 6 degrees of freedom. We assume that we are in the general case where, a priori, all robots can manipulate all objects. We also assume that the robots cannot grasp more than one object at a time (but the scheme can be extended). We name $CS_{R_i}(0 < i \leq r)$ and $CS_{M_j}(0 < j \leq m)$ the configuration spaces of the robots and the objects respectively. The composite Configuration Space (i.e. the global space) is $CS = CS_{R_1} \times ... \times CS_{R_r} \times CS_{M_1} \times ... \times CS_{M_m}$.

Definition 12: the manipulation constraints.

The manipulation constraints are:

- *Definition of sets of allowed stable configurations for objects. We define subsets for each $CS_{M_j}(0 < j \leq m)$ where the objects can be placed in stable position.*

- *For each robot-object pair, we define a set of relative positions that correspond to valid grasps.*
- *A grasp remains fixed when the robot transports the object.*

These definition account for continuous grasps and placements (Siméon et al. (2003)). Note that other means of object manipulation based on a similar formalism have been proposed such as pushing (Stilman and Kuffner (2004, 2006), Lynch and Mason (1996)), dexterous multi-finger manipulation (Saut et al. (2007)) or pivoting bulky objects (Yoshida et al. (2007)) introduce other specific constraints that we do not use here. Besides, in order to simplify the discussion, we add another constraint: only one robot moves at a time.

Definition 13: Subsets of CS

We consider the following subspaces of CS:

- $CP(R_i)(0 < i \leq r)$*, the set of configurations where robot R_i is not attached to any object and can move freely while all the objects are in stable positions or are grasped by another robot.*
- $CP(M_j)(0 < i \leq m)$*, the set of configurations where the object M_i is in a stable position while the other objects are in a stable position or are grasped by a robot.*
- $CG(R_i.M_j)(0 < i \leq r)(0 < j \leq m)$*, the set of configurations where R_i grasps object M_j following an allowed relative position (see Definition 12) and where the other objects are in stable positions or grasped by another robot.*

Theorem 14: Solution of a multi-robot manipulation problem

*A **solution for a multi-robot manipulation problem** is a path in CS_{free} constrained by the manipulation constraints. It is a finite sequence of **transit** R_i paths and **transfer** $R_i.M_j$ paths. Transit paths and transfer paths involving the same robot R_i are separated by grasp/ungrasp operations.*

A demonstration can be found in Siméon et al. (2003). We give hereafter the definition of transit and transfer paths.

Definition 15: Transit and Transfer

*A **transit path for** R_i is a path where the robot moves alone and where other objects and robots remain static. This path belongs to $CP(R_i)$. Note that any path in $CP(R_i)$ is not necessarily a transit path. Indeed, numerous paths in $CP(R_i)$ involve other robots or objects motions. A **transfer path for** $R_i.M_j$ is a path where the robot R_i moves while maintaining a fixed grasp of the object M_j. Transfer $R_i.M_j$ paths belong to $CG(R_i.M_j)$. Note that, as with transit paths, not any path in $CG(R_i.M_j)$ is a valid transfer path.*

Note that $CP_(R_i) \cap CG(R_i.M_j)(0 < i \leq r)(0 < j \leq m)$ are interesting subspaces. They correspond to configurations where R_i transit paths and $R_i.M_j$ transfer paths can be connected by grasp/ungrasp operations. Indeed, a $CG(R_i.M_j)$ configuration where M_j can be released in a stable configuration is a $CP_(R_i)$ configuration where the robot and the object are not yet attached.

These definitions are illustrated by figure 3, which represents a problem where an object named $cbox$ must be transferred by a robot named $forklift$. We represent the different subspaces composing CS_{free}. One can see that $CP(forklift)$ and $CG(forklift.cbox)$ have several connected components. $CG(forklift.cbox)$ is composed of two connected components induced by the two possible grasps illustrated in the bottom of the figure. $CP(forklift)$ is composed of three connected components. Only one of them (numbered 3) intersects both components of $CG(forklift.cbox)$. Consequently, a solution path must involve a re-grasping operation. $forklift$ must put $cbox$ in an intermediate configuration where it can change from one grasp to the other with a transit path included in the connected component 3 before bringing the object $cbox$ to its final configuration.

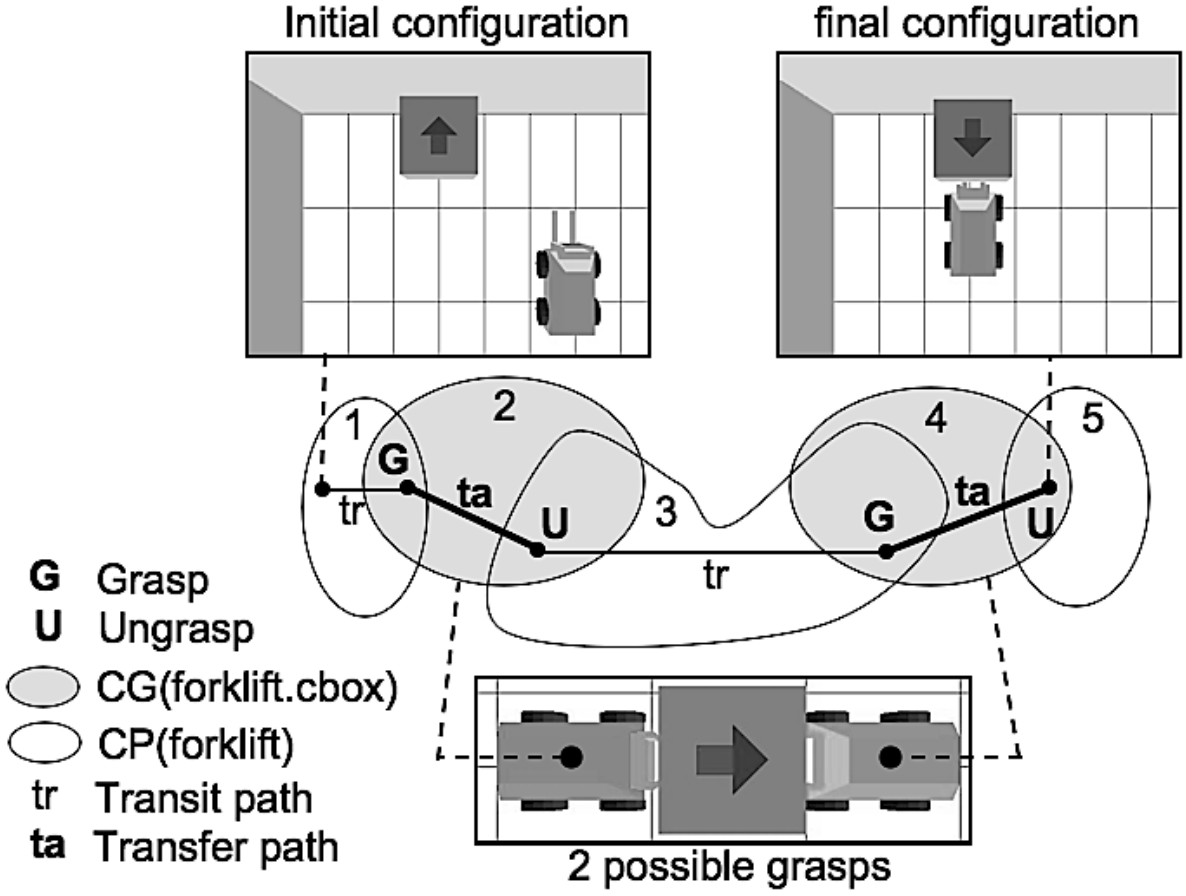

Figure 3: Illustrations of subspaces and the manipulation definitions in a little problem

The structure of CS for a multi-robot manipulation problem is going to guide us for the definition of $\mathcal{R}$ relation for a given problem. We will see that in our problem definition, an action will involve a set of transit paths, a set of transfer paths set, a grasp or ungrasp action or no motion.

3.2 An example of the definition of a multi-robot manipulation problem with symbolic constraints

Now we can present, through the radio-switch problem, how we precisely define a problem based definition 11. The main point here is to show how to build a $\mathcal{R}$ relation (definition 6) by taking into account the structure of CS previously discussed.

Note that the method is general. We will give in §5.1 some clues on how to partially automatize the definition of a given problem.

For the radio-switch problem, the $PLACE$ set is composed of the following propositions:

- P_FORKLIFT this proposition means that $forklift$ is not attached (i.e. not grasping) to any object. The other robots and objects can be anywhere.
  We define (P_FORKLIFT, $CP(forklift)) \in \mathcal{R}$
- P_FORKLIFT_CG_CANGRASP_CBOX this proposition means $forklift$ can apply a grasp operation on $cbox$. In other words, the relative positions and orientations of $forklift$ and $cbox$ correspond to an allowed grasp.
  We define (P_FORKLIFT_CG_CANGRASP_CBOX, $CP(forklift) \cap CG(forklift.cbox)) \in \mathcal{R}$
- P_ARMBOT this proposition means $armbot$ is not attached to any object.
  We define (P_ARMBOT, $CP(armbot)) \in \mathcal{R}$
- P_FORKLIFT.CBOX this proposition means that $forklift$ and $cbox$ are attached.
  We define (P_FORKLIFT.CBOX, $CG(forklift.cbox)) \in \mathcal{R}$
- P_FORKLIFT.CBOX_CANUNGRASP this proposition means that the robot $forklift.cbox$ can apply an ungrasp operation to form again two independent entities: $forklift$ and $cbox$. In other words, $cbox$ is attached to $forklift$ and is in a stable placement.
  We define (P_FORKLIFT.CBOX_CANUNGRASP, $CP(forklift) \cap CG(forklift.cbox)) \in \mathcal{R}$
- P_CBOX This proposition means $cbox$ is somewhere on a stable placement. It is not attached to any robot.
  We define (P_CBOX, $CP(cbox)) \in \mathcal{R}$

For a given problem, we add propositions that specify the initial geometric situation as well as the goal situation.

- P_FORKLIFT_INIT means $forklift$ is on its initial place. We define (P_FORKLIFT_INIT, $c0) \in \mathcal{R}$ where $c0 \in CP(forklift)$.
- P_CBOX_INIT and P_CBOX_GOAL are the propositions which specify initial and goal placements for $cbox$. We define (P_CBOX_INIT, $c1) \in \mathcal{R}$ and (P_CBOX_GOAL, $c2) \in \mathcal{R}$ where $c1$ and $c2$ are two configurations which belong to $CP(cbox)$. Note that the goal placement can be specified as a subset of $CP(cbox)$.

There are also propositions that allow to specify the geometric pre-conditions that have to be satisfied in order to apply symbolic actions i.e. actions that have no effect on the geometry.

- P_FORKLIFT.CBOX_GOOD_RECEPTION the place where the composed robot $forklift.cbox$ is in a good reception area. We define (PFORKLIFT.CBOX_GOOD_RECEPTION, $CS0$) $\in \mathcal{R}$ where $CS0$ is a subset of configurations of $CG(forklift.cbox)$. $cbox$ is used as by the robot as an antenna. In our implementation $CS0$ is delimited by two constraints on the position of $cbox$.
- P_ARMBOT_SWITCH the place where $armbot$ should be placed in order to be able to push the switch. We define (P_ARMBOT_SWITCH, $CS1$) $\in \mathcal{R}$ where $CS1 \subset CP(armbot)$ and represents a set of configurations where the end effector of $armbot$ touches the switch.

We also define the following propositions which do no belong to $PLACE$:

- SWITCH_PUSHED means that the switch has been pushed.
- DATA_RECEIVED means that the data has been received by wireless connection.

Now we can build a set of planning operators - also called actions in STRIPS-like (Fikes and Nilsson (1971)) vocabulary - that comply with the manipulation constraints. We will not define here all the operators but only some examples.

The GOTO_FORKLIFT_CANGRASP_CBOX operator means that $forklift$ moves to reach a configuration where it can grasp $cbox$.

```
(:action GOTO_FORKLIFT_CANGRASP_CBOX
 :precondition
        P_FORKLIFT
 :effect
        not P_FORKLIFT
        P_FORKLIFT_CANGRASP_CBOX)
```

The GOTO_ARMBOT_SWITCH operator means that $armbot$ moves in order to reach a configuration where its end effector touches the switch.

```
(:action GOTO_ARMBOT_SWITCH
 :precondition
        P_ARMBOT
 :effect
        not P_ARMBOT
        P_ARMBOT_SWITCH
        SWITCH_PUSHED)
```

Each GOTO-XXX operator corresponds to a set[3] of transit or a transfer paths in CS. In our problems, we also need GOTO-XXX operators to define the motions from the initial configurations and from/to

3. can be infinite

specific places like P_ARMBOT_SWITCH to more general ones like P_ARMBOT. This last aspect lets the planner move the robot even if there is no "symbolic interest" to do it. This is the case when it is necessary to find an intermediate position, for instance, to give room to another robot or to perform a regrasp sequence.

GRASP_FORKLIFT_CBOX is the robot composition operator for $forklift$ and $cbox$. UNGRASP_FORKLIFT_CBOX, the decomposition operator, can be defined symmetrically.

```
(:action GRASP_FORKLIFT_CBOX
 :precondition
        P_FORKLIFT_CANGRASP_CBOX
        P_CBOX
 :effect
        not P_FORKLIFT_CANGRASP_CBOX
        not P_CBOX
        P_FORKLIFT.CBOX_CANUNGRASP)
```

We also need to define GRASP-XXX and UNGRASP-XXX operators for the initial and goal configurations of $cbox$. Note that these operators do not need any motion. In other words, the configuration of the composite mechanical system is unchanged. Only the set of compositions of robots and objects is changed.

Besides operators which have consequences on the geometric world, we define purely symbolic operators. For instance:

```
(:action RECEIVE_DATA
 :precondition
        P_FORKLIFT.CBOX_GOOD_RECEPTION
 :effect
        DATA_RECEIVED)
```

This is typically an action that has a geometric placement constraint expressed by the P_FORKLIFT.CBOX_GOOD_RECEPTION proposition. It is of course possible to define operators that have no link (no precondition and no effect) with the geometric environment.

We can also overload the GOTO-XXX, GRASP-XXX or UNGRASP-XXX operators. For instance, we can add a predicate MOTOR_LIFT_OK as a precondition to GRASP operators involving a given robot.

To complete the description of the *radio-switch* we just need to give an initial state $s0$ and a goal g expressed as a conjunction of propositions (definition 11). Here we have:

$s0$ = (P_FORKLIFT_INIT $\wedge$ P_ARMBOT_INIT $\wedge$ P_CBOX_INIT)

g = (DATA_RECEIVED $\wedge$ P_CBOX_GOAL)

$s0$ means $forklift$, $armbot$ and $cbox$ are in their initial positions, the data have not been received and the switch has not been pushed (we work in the Close World Assumption). g means the data have been received and $cbox$ is in its final configuration. As one can see, the initial state is

completely defined while the goal state is only partially specified. We do not care about the state of the other robots and objects.

It is worth noting that this representation does not need to be changed for problems belonging to the same class. Only the $\mathcal{R}$ relation and/or the geometry of the fixed environment (or of the robots and objects) have to be modified to create new instances of a problem class.

4. Probabilistic approaches to explore CS and consequences

A strong link is now effective between the symbolic and the geometric world descriptions. Moreover, we have defined a search space for a task planner. To be able to navigate in this search space, we should devise means to validate states (10-2), and to find configurations in specific subsets of CS_{free}, i.e. the sets described in (10-5).

It has been shown that the best exact algorithm for the classical motion planning problem is exponential in the number of the degrees of freedom of the system. Probabilistic approaches (Barraquand and Latombe (1991), Kavraki and Svestka (1996), LaValle (1998)) are well suited to deal with motion planning in high dimensional spaces. For us, even apparently simple problems like the radio-switch problem are highly dimensional: 19 dimensions in this case. Nevertheless, even with a probabilistic approach the number of dimensions remains high. We introduce a two-steps algorithm to break the complexity of the search in CS. The first step is based on the use of several roadmaps which focus on a limited number of degrees of freedom without taking into account the other dimensions of CS. The role of the second step is to validate the paths found in the first step, but this time, by taking into account the other dimensions. As this second step can be very costly, the method will develop a strategy that limits its use as much as possible.

4.1 The use of several roadmaps

Basically, probabilistic approaches capture the free configuration space topology by building graphs called *roadmaps*. A roadmap node represents a configuration and an arc represents a collision-free motion.

In our case, and in order to limit the complexity, we build several roadmaps corresponding to the subspaces mentioned in section 3 (see Gravot et al. (2002) and Gravot and Alami (2003)).

In figure 4, we represent the roadmaps that have been developed for the *radio-switch* problem. The $forklift$ transit roadmap captures $CP(forklift)$, the $cbox$ placement roadmap captures $CP(cbox)$, The $forklift.cbox$ transfer roadmap captures $CG(forklift.cbox)$ and the $armbot$ transit roadmap captures $CP(armbot)$.

As we will see below, the roadmaps can be constructed incrementally. The illustrated roadmaps have a limited number of nodes but are however sufficient to solve the problem. The links between roadmaps are represented by dashed lines. They correspond to configurations where grasp/ungrasp actions (and the associated composition/decomposition of robots and objects) can be performed. These configurations are obtained by using direct or inverse kinematics.

Note that in order to reduce the dimensions of the search space, the roadmaps are built independently, one for each robot, movable object or composed robot. In other words, we build the

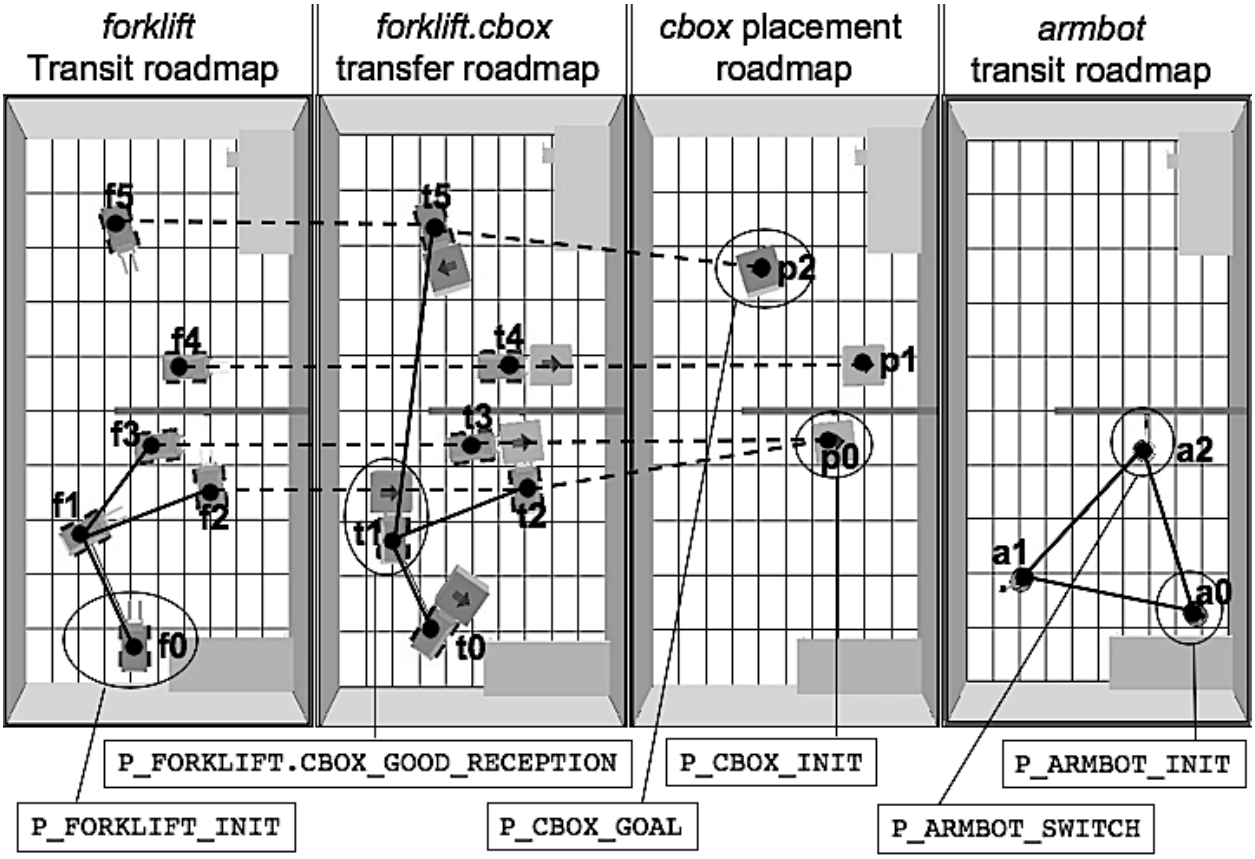

Figure 4: Roadmaps developed for the *radio-switch* problem. We also indicate the considered $place$ propositions.

roadmaps by taking into account only the static environment. Consequently, arcs or nodes will be valid in some "geometric contexts" and invalid in other ones. With this method, a configuration of CS is given by composing a set of nodes belonging to different roadmaps.

4.2 Advanced motion planning techniques

Continuous grasp and complex manipulations We can define not only a discrete set of grasping configurations but also continuous grasps. For example, the cylinders of the Geometric Hanoi Tower Problem (§6.2) can be grasped all around them. In this case, it is interesting to explore more intensively the $CP(R_i) \cap CG(R_i.M_j)$ spaces. Specific roadmaps can be used to capture the topology of these spaces and to derive solutions to intricate manipulation problems with several regrasping. These roadmaps can be naturally integrated in our "multi-roadmap" approach. The

reader can refer to Alami et al. (1994) and to Siméon et al. (2003) for a detailed discussion on manipulation problems in presence of continuous grasps and placements.

The Relative Roadmaps. We have introduced the notion of Relative Roadmap in order to avoid to "solve" several times a very similar sub-problem consisting in studying the approach a given object by a given robot until contact on near-contact, in several contexts. To build a Relative Roadmap, we place the involved robot in an environment containing only one obstacle: the object to grasp. The construction of the Relative Roadmaps consists in finding paths reaching the grasp configurations from configurations where we assume that the robot is far enough from the object (see figure 5). We build Relative Roadmaps with a RRT algorithm which has proved to be more efficient for this kind of constrained motion (LaValle (1998)).

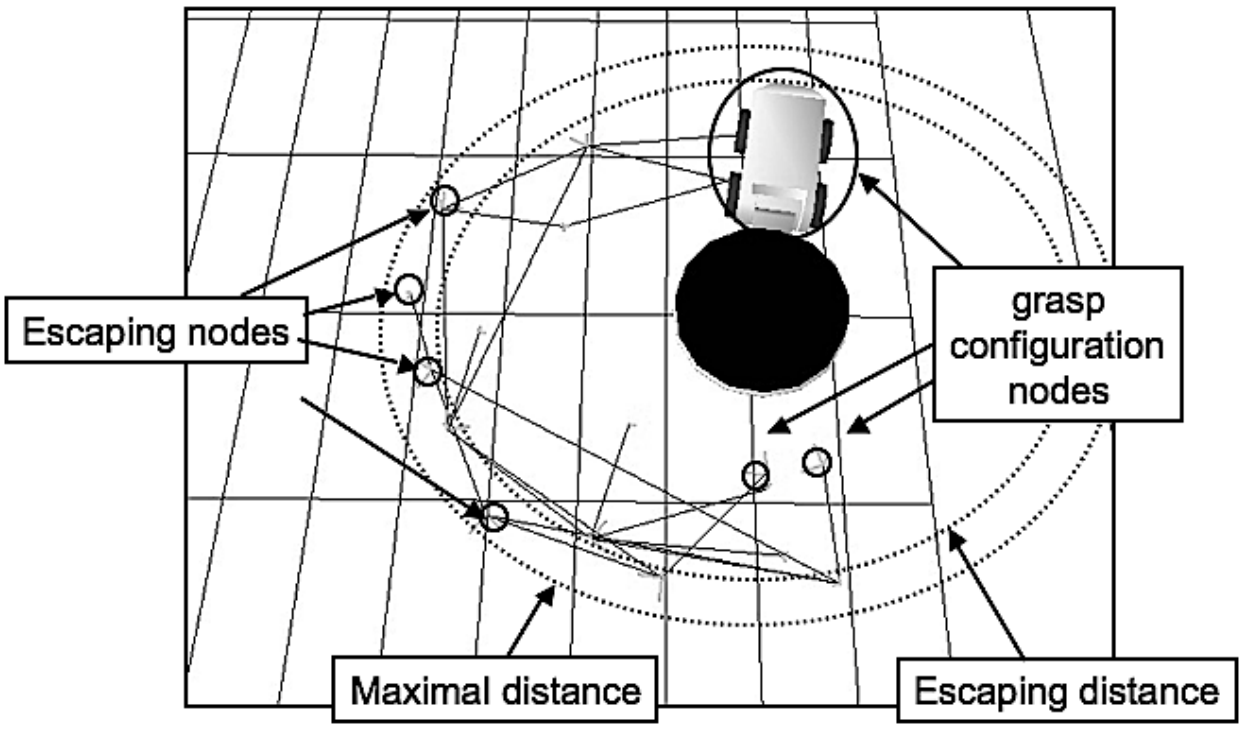

Figure 5: The Relative Roadmap that encodes several approach and contact paths of $forklift$ relatively to a cylindrical object.

Whenever we have to deal with the problem of the "docking" of one robot to one object, we use a pre-computed relative roadmap by "copying" its edges and nodes in the current environment. "Copying" a configuration means to apply a geometric transformation (translation, rotation) on some degree of freedom. This techniques is very close to the "kinematic roadmap" method proposed in LaValle et al. (1999).

4.2.1 The implemented $\mathcal{R}$ relation and actions applicability

The implemented $\mathcal{R}$ relation. With the roadmaps method, each $place$ proposition relates to a set of nodes of a specific roadmap. We associate to each place a method to generate such nodes. For example:

- P_FORKLIFT is in relation with all the nodes of the $forklift$ transit roadmap. The method to generate such nodes can be a general method to add nodes in standard motion roadmaps. In our implementation we use the PRM visibility method described in Siméon et al. (2000).
- P_FORKLIFT_CANGRASP_CBOX is in relation with the set of nodes in the $forklift$ transit roadmap where $forklift$ can grasp $cbox$. A way to generate such nodes is to select one node from the $cbox$ placement roadmap and to build a node into the $forklift$ transit roadmap by inverse kinematics. Note that these nodes are also valid in the $forklift.cbox$ transfer roadmap and are added in it. They can be related to the P_FORKLIFT.CBOX_CANUNGRASP proposition.
- P_FORKLIFT.CBOX_GOOD_RECEPTION is in relation with a subset of nodes in $forklift.cbox$ transfer roadmap. We build those nodes by sampling them in a specific region represented by a rectangular area in figure 1.

Action application.

Here we explain how we use the roadmaps to solve problems from definition 11. The main point is to find at least one configuration belonging to $cs_{s'} = cs_s^{reach} \cap cs_{s'}^{def}$ (10-5) where s' is a new state obtained through an application of action a.

We once again rely on the radio-switch problem example. Let us imagine that the initial state $s0 = (sym_{s0}, cs_{s0})$ such that

- sym_{s0} = (P_FORKLIFT_INIT $\wedge$ P_ARMBOT_INIT $\wedge$ P_CBOX_INIT)
- $cs_{s0} = \{((ti0), (p0), (a0))\}$.

Let us imagine now that we want to apply the action:

```
(:action GOTO_FORKLIFT_INIT_CANGRASP_CBOX
 :precondition
  P_FORKLIFT_INIT
 :effect
  not P_FORKLIFT_INIT
  P_FORKLIFT_CANGRASP_CBOX)
```

This action can be applied because it satisfies the conditions of action application (10-1) and (10-2). This action results in a new state $s1 = (sym_{s1}, cs_{s1})$ with, by application of (10-3):

- sym_{s1} = (P_FORKLIFT_CANGRASP_CBOX $\wedge$ P_ARMBOT_INIT $\wedge$ P_CBOX_INIT)

cs_{s1}^{def} is the set of valid configuration included in the intersection of the subspaces defined by the places of the state. The use of roadmaps gives us a finite set of configurations belonging to this intersection. The proposition
P_FORKLIFT_CANGRASP_CBOX denotes all the configurations where $forklift$ is in contact with $cbox$ such that that they can form a composed robot $forklift.cbox$. Moreover, the proposition P_CBOX_INIT and P_ARMBOT_INIT indicate that the object $cbox$ and the robot $armbot$ is in their initial configurations. $p0$ and $a0$. In the current state of the computation, the configurations of cs_{s1}^{def} are:

((f2),(p0),(a0)) and ((f3),(p0),(a0)).

cs_{s0}^{reach} are the reachable configurations from cs_{s0}. For the implemented reachability function, these configurations belong to the connected components in the roadmaps corresponding to robot motions. The connected components of $(f0)$ and $(a0)$ contains respectively 4 nodes 3 nodes. Consequently, we obtain 12 configurations for cs_{s0}^{reach}.

The intersection of cs_{s1}^{def} and cs_{s0}^{reach} gives us two candidates $((f2),(p0),(a0))$ and $((f3),(p0),(a0))$. Please note that at this step of the computation, these configurations are just candidates. We have to ensure that they are effectively reachable without collision from the initial configuration. It may be possible, for example, that the local path from $(f0)$ to $(f2)$ is in collision with $(p0)$[4].

As already mentioned, we need a second computation step to select configurations candidates for cs_{s1}. This second step is called validation. If we succeed in *validating* at least a configuration, we obtain $cs_{s1} \neq \emptyset$ which means that $s1$ becomes a valid state that can be selected to explore more deeply the search space.

The validation algorithms, explained in the next section, can be costly. Indeed, we work on combinations of nodes, which is exponential in the number of robots and movable objects. As we explore CS hierarchically, the possible collisions with the static obstacles have already been checked by the roadmaps construction. It remains to consider all the possible interactions between robots and movable objects in the environment.

5. Interweaving task planning process and motion planning process

Our planner is based on a hybrid planning process in which a fast heuristic forward planner for "high-level" plans is used in close interaction with a manipulation planning subsystem for instantiating and validating actions in the 3D environment.

5.1 About the implemented problem formulation

For simplicity, we have introduced our definitions on the basis of a set-theoretic representation for task planning. In our implemented planner, we employ a more elaborate task planning representation based on PDDL 2.1 language (Fox and Long (2003)). It allows us mainly to use first-order literals and logical connectives instead of propositions, and consequently to write more concise planning problems descriptions. For instance, in our problem we need to define only one parametrized GOTO operator.

4. The use of the Relative Roadmaps should help, as mentioned earlier, to solve this sub-problem.

Another important remark relates to the fact that it is possible to define a systematic way to build hybrid - symbolic and geometric - planning problems. Indeed, we have greatly automatized the production. All the symbols and the implemented $\mathcal{R}$ relation can be automatically generated based on a compact description of the robots, the objects, and of the allowed compositions together with the specific places needed by the problem class. For instance, in the radio-switch problem we declare:

```
SYMBOLS:
ROBOTS:{FORKLIFT; ARMBOT}
OBJECTS:{CBOX}
COMPOSITION:{FORKLIFT.CBOX: FORKLIFT + CBOX}
SUBSPACES:{FORKLIFT CP ;ARMBOT CP ;
          CBOX CP ; FORKLIFT.CBOX CG}
LINK:{FORKLIFT CP / FORKLIFT.CBOX CG ;
      CBOX CP / FORKLIFT.CBOX CG}
ZONES:{FORKLIFT.CBOX CG: GOOD_RECEPTION
      ARMBOT CP: SWITCH}
INIT:{FORKLIFT CP ;CBOX CP; ARMBOT CP}
GOAL:{CBOX CP}

METHODS:
(FORKLIFT, CP, ‘‘forklift transit’’)
(ARMBOT,  CP, ‘‘armbot transit’’)
(CBOX,  CP, ‘‘cbox placement’’)
(FORKLIFT.CBOX, CG, ‘‘forklift.cbox transfer’’)
(ARMBOT, SWITCH, ‘‘method_1’’)
(FORKLIFT.CBOX, GOOD_RECEPTION, ‘‘method_2’’)
```

We also provide a geometric description of the named robots and movable objects and of the fixed obstacles.

Based on this description, PDDL files are generated that contain all the needed symbols as well as GRASP, GOTO, and UNGRASP operators. A file describing the $\mathcal{R}$ relation is also generated. Some methods to generate nodes for $place$ propositions can be inferred. The user has to define the other operators (ex: RECEIVE_DATA) as well as the specific methods that will be used by the planner to generate relevant configuration nodes (e.g. the nodes corresponding to GOOD_RECEPTION).

5.2 Software Architecture

Our planner, named aSyMov[5], integrates and extends (see Cambon et al. (2004)) a task planner called Metric-FF (Hoffmann (2003)) and a set of motion and manipulation planning functions based on Move3D (Siméon et al. (2001)). Figure 6 presents aSyMov software architecture.

The input of the planner consists in:

5. aSyMov: a Symbolic Move3D

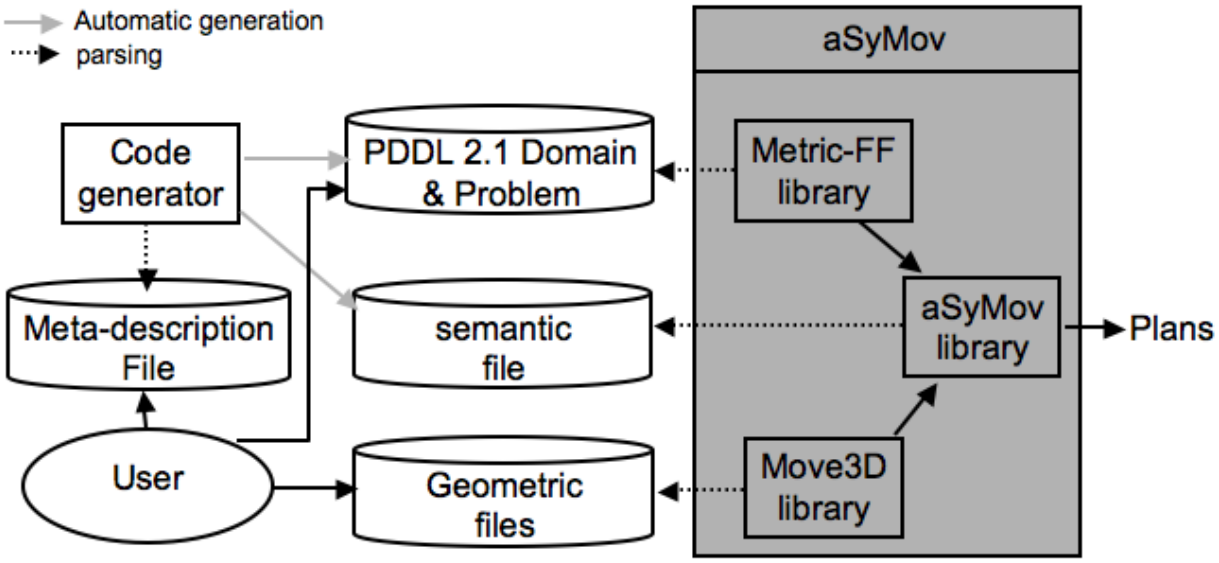

Figure 6: The aSyMov software architecture.

- a **symbolic data** file including a PDDL description of the symbolic domain and problem.
- a **geometric data** file containing geometric representations of the static obstacles, robots and objects as well as the robots kinematics and the associated roadmap expansion methods.
- a **semantic data** file that encodes the $\mathcal{R}$ relation.

5.3 The planner algorithms

A state s in aSyMov is represented by a data structure composed of:

- the symbolic state sym_s (the state as it can be used a purely symbolic task planner).
- a set of configurations proven to belong to cs_s (if this set in not empty, the state s is valid).
- a set of configurations that are candidate to belong to cs_s. We name this set cc_s. An example of computation of this set is given in §4.2.1. These configurations are not validated by the second step of validation.

The aSyMov overall procedure is illustrated in figure 7.

The loading consists in the creation of the data structures for Metric-FF and Move3D. The preprocessing consists in a symbolic domain analysis for Metric-FF. It also instantiates all the potential actions. On the other side, Move3D creates bounding boxes to speedup the collision checker computation. aSyMov then relates the structures created by the two planners on the basis of the semantic file. It checks the coherence of the data, the correspondence between the symbols and

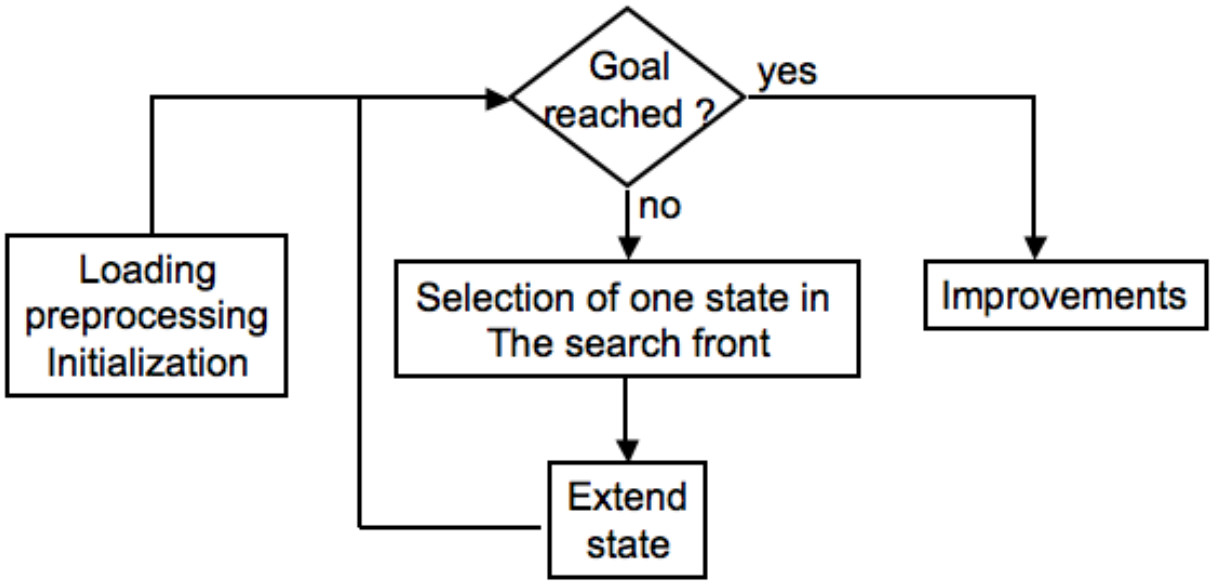

Figure 7: The aSyMov overall algorithm.

the roadmaps, the validity of the initial state or the non-triviality of the associated task planning problem, etc. Finally the initial state is created and added to the front search.

When a plan is found, aSyMov performs a set of improvement processes. They involve standard optimization of the planned trajectories and parallelization of the plan in case this one involves more than one robot. The parallelization is a two steps algorithm that first parallelizes the symbolic plan, using a technique described in Regnier and Fade (1991), and coordinates the trajectories with a method close to the one described in O'Donnell and Lozano-Perez (1989).

Note that, after initialization, all the needed roadmaps are created but are just filled with the initial and final configurations. So at each step of the search process, aSyMov will try to arbitrate between finding a plan with the level of knowledge already acquired, or "investing more" in a deeper knowledge of the topology of the different configuration subspaces it manipulates. Indeed, we have decided to represent the expansion of the roadmaps at the same level as the actions.

The core algorithm is the "Extend State" procedure illustrated in figure 8. This algorithm tries to add a new state $s' = (sym_{s'}, cs_{s'})$ by the application of actions on a state $s = (sym_s, cs_s)$. After this tentative, an update of the front search is performed by changing the cost of the front search elements or/and by adding the new state. Then the "best" state of the front search is selected and the process continue until a state is found that satisfies the goal.

Hereafter, we give details on the Extend State procedure.

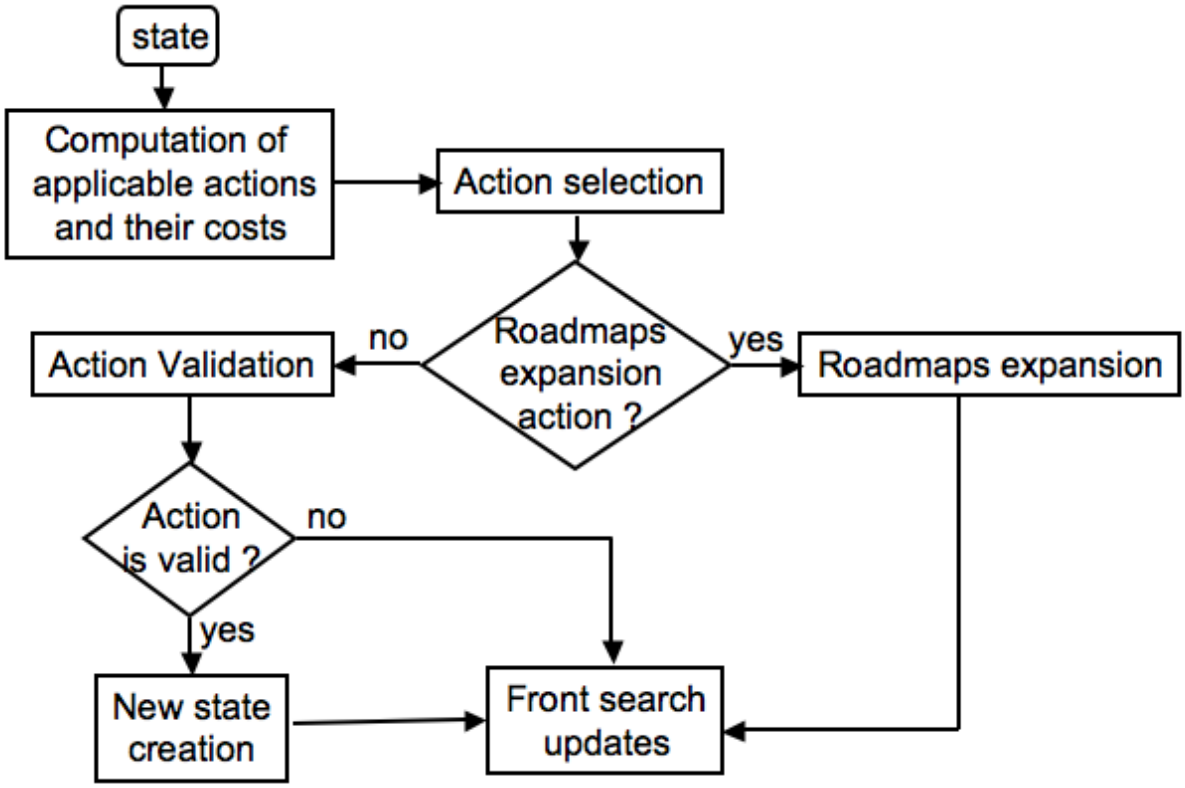

Figure 8: The Extend State algorithm.

5.4 Actions, cost of actions and cost of states

5.4.1 Determining applicable actions

A valid (i.e. $cs_s \neq \emptyset$) state s is chosen . The ***applicable actions*** are those which satisfy (10-1), i.e. actions which can be applied to the symbolic state sym_s. We define a second type of actions, the ***roadmaps expansion actions***, which consist in spending some computation time trying to expand the current roadmaps. Each applicable action has an associated roadmaps expansion action. For example, in the radio-switch problem, from the state $s0$ defined in §4.2.1, we can choose between the following actions:

- `GOTO_FORKLIFT_INIT_CANGRASP_CBOX`: the action where $forklift$ goes from its initial configuration to a place where it can grasp $cbox$.
- `GOTO_FORKLIFT_INIT_ANYWHERE`: the action where $forklift$ goes from its initial configuration to another place. This kind of action can be useful to position a robot on an intermediate configuration (for example, to leave room for another robot to pass).
- `GOTO_ARMBOT_INIT_SWITCH`: the action where $armbot$ goes from its initial configuration to a place where it can push the switch.

- GOTO_ARMBOT_INIT_ANYWHERE: the action where $armbot$ goes from its initial configuration to another place.
- EXPAND_GOTO_FORKLIFT_INIT_CANGRASP_CBOX: the action which adds nodes to the $cbox$ placement roadmap and then constructs nodes by inverse kinematics in the $forklift.cbox$ transfer roadmap. These last nodes are also added in the $forklift$ transit roadmap to create links between roadmaps.
- EXPAND_GOTO_FORKLIFT_INIT_ANYWHERE: the action which adds nodes to the $forklift$ transit roadmap.
- EXPAND_GOTO_ARMBOT_INIT_SWITCH: the action which generates configurations by inverse kinematic to obtain nodes in the $armbot$ transit roadmaps where the $armbot$ gripper touches the switch.
- EXPAND_GOTO_ARMBOT_INIT_ANYWHERE: the action which adds nodes to the $armbot$ transit roadmap.

5.4.2 Selecting an action

Action selection is based on a cost estimation process. We recall that the symbolic formulation of the problem, i.e. the problem without taking into account (10-2) and (10-5) is a relaxed instance of the overall problem.

For each applicable action a the process is:

- **1st step.** The planner computes the set of candidate configurations cc_s for the resulting state. If $cc_s = \emptyset$, a is removed from the selection. This action has no chance of being valid.
- **2nd step.** The planner calls the symbolic task planner to compute a symbolic plan where the initial state is $sym_{s'}$[6]. If a symbolic plan is found, its length gives us a first *heuristic cost* value for a. Note that if we choose an optimal task planner, this heuristic is admissible. Note that it is only here that we use the symbolic task planner. In fact, it is used only to compute the heuristic.
- **3rd step.** The planner propagates the set of candidate configurations through the symbolic plan found in the 2nd step. If there are candidate configurations for each symbolic state traversed, the heuristic cost keeps the value given by the second step. If this is not the case, we increase the cost. This lets us make the heuristic cost take into account the current knowledge of the geometry of the environment.

The total cost of an action is: *total cost* $=$ *Heuristic cost* $+$ *Cost of number of failures*

The *heuristic costs* are those computed in the 3rd step. For the roadmaps expansion actions, this cost is the same as for their associated applicable action.

The *Cost of the number of failures* allows us to avoid trying to validate the same action too often. It is possible that there are no motions corresponding to an action. This cost is computed as a

6. In our implementation, a lazy evaluation mechanism allows to compute only once a symbolic plan starting at $sym_{s'}$

linear function of the number of failures in the attempts to validate this action. For the roadmaps expansion action, this cost limits the number of nodes added for a given place. This cost is then a linear function of the number of selections of this action.

Finally, the planner selects the action with the lowest cost.

Important remark. It is mainly through the selection of the next action that the link between the symbolic world and its geometric counterpart is exploited. A symbolic plan is computed and serves as a first basis to our heuristic. Then, the propagation of roadmaps nodes through the candidate configurations propagation gives an idea about how the geometry is explored: this is the second basis of our heuristic. Finally the failures cost give us an idea about the feasibility of an action: this is the third basis of our heuristic. It is also at the selection of an action that we decide if a roadmap expansion is worth to be performed.

5.4.3 Selecting states

Each search step begins by a state selection (see figure 7). At each step we update the front search by giving costs to states. Thus we can compare them and choose the cheapest. The cost of a state is:

State cost $=$ *Accumulated cost* $+$ *Heuristic cost* $+$ *Cost of the Number of failures*

where:

- The *Accumulated cost* is the cost to reach the state from the initial state expressed in terms of number of actions.
- The *heuristic cost* is the sum of the non-validated applicable actions costs of this state.
- The *Cost of the Number of failures* is obtained with a linear function of the number of failures of the Extend State algorithm for this state.

We select states from the front search with a probabilistic weighted selection.

Remark. Our algorithm for the exploration of the state space is similar the A* algorithm. The main difference is that we can never delete simply an action or a state from the front search. Indeed, the use of probabilistic methods cannot ensure that there is no solution to a motion planning sub-problem. To limit the undesirable effects of a constantly growing front search, we introduced the failure costs.

5.5 Applying an action

Applying an action means trying to validate an applicable action or enriching the roadmaps in the case of the roadmaps expansion actions.

5.5.1 Validate an applicable action

Validating an action $a : (sym_s, cs_s) \rightarrow (sym_{s'}, cs_{s'})$ means finding at least one configuration $c \in cs_{s'}$. This is performed through a constraint propagation mechanism along the current plan - i.e.

the sequence of actions from the initial state to (sym_s, cs_s). This mechanism is allowed to change the geometric instantiations of the preceding actions. The process is lazy, it stops at the first configuration found and the resulting state becomes valid. However, if the action is not validated, it is not deleted it from the front search. Indeed, later on, a *roadmaps expansion* action may add nodes which will allow the planner to try again to validate the action. This is an important feature of the planner, with a risk of inefficiency, but it is crucial since it "provides" a certain level of completeness. More interestingly, it opens the possibility to arbitrate between finding a new plan with the level of knowledge already acquired, or exploring more deeply different sub-manifolds of CS_{free}.

The question here is the management of the set of the candidate configurations cc_s for each state. When a state is s' created, $cc_{s'}$ is initialized not only on the basis of the precedent cs_s but also the precedent cc_s. Each configuration $c \in cc_s$ maintains a tree that represents the ***explanation*** from where it comes. Figure 9 illustrates the explanation of a candidate configuration $C4 \in cc_{s4}$ the 4th state of a plan under construction. Some of the configurations which belong to the explanation of C4 are valid and others are just candidates. For a candidate configuration c, we call ***level of a candidate configuration*** the number of states that have to be traversed backward its explanation tree before reaching at least one valid configuration. In our example figure 9, the level of C4 is 1 and the level of C2-0 is 2.

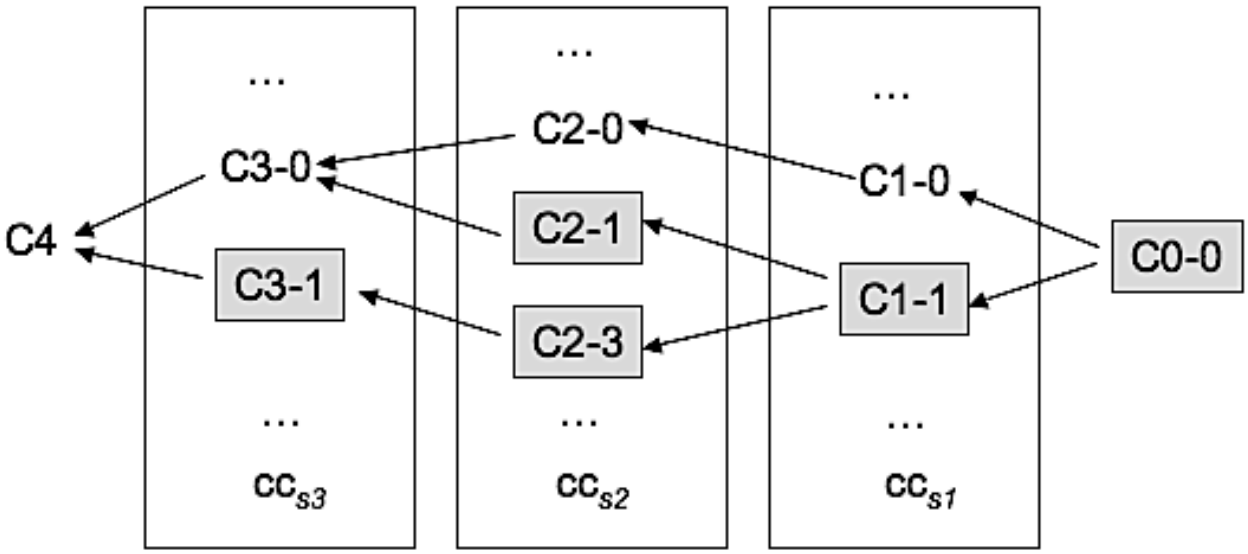

Figure 9: Example of an explanation tree for a candidate configuration C4. The valid configurations are emphasized by grey rectangles.

The Validation algorithm has two arguments: the state `s` and the length `(n)` of the plan to reach `s`,

```
Validate(s,n)
```

```
1  CC_s = ComputeCandidate(s);
2  current_level = 1;
3  valid = false;
4  while (current_level <= n)
5     CandidateOfLevel = ConfigurationsOfLevel(CC_s, current_level)
6     for each C in CandidateOfLevel do
7        Exp = ComputeExplanation(c);
8        valid = TestAllPathsExplanation(Exp);
9        if (valid = true) then
10           CS_S = CS_S + C;
11           return true
12       end if;
13    end for;
14    current_level = current_level + 1;
15 end while
16 return false;
```

The procedure `ComputeCandidate` has been explained in §4.2.1. This computation is done not only on the basis cs_s and the cc_s of the preceding state. The procedure `ConfigurationsOfLevel` builds a set containing all the configurations of cc_s of a given level. The planner tries to validate the higher level candidates first. They need less computation.

The procedure `ComputeExplanation` extracts the explanation (see figure 9) of a configuration `C`. In our implementation, this procedure is done during the `ComputeCandidate` algorithm by maintaining lists. The procedure `TestAllPathsExplanation` tests iteratively the paths of an explanation. When it finds a valid path, the procedure returns the boolean value `true`. The procedure checks whether the list of paths between two configurations is collision-free. We recall that with our multi-roadmaps approach, this test is done taking into account all the other robots and objects. A more graphical explanation of this algorithm can be found in Gravot et al. (2003). The strategy of `TestAllPathsExplanation` first tries to validate configuration `C` by the simplest way. In our example the procedure will try first to validate the path between C3-1 and C4. If this path is not valid, it will try to validate the path between C2-1 and C3-0 and then between C3-0 and C4, etc. During this procedure, the cs_s part of a preceding state can be updated. For example, if the planner validates the path between C2-1 and C3-0 then C3-0 is added in the cs_{s3} set.

The Validation algorithm is stopped when a configuration found and added to the cs_s part of the state. If no configuration is found, the state s' is not validated with the level of information that the system has currently on the topology of CS. Action a is not removed from the front search, further roadmaps expansion actions may make s' valid. The explanations are updated whenever a node is added in the roadmaps.

The Validation algorithm is intrinsically costly. In the worst case, it is exponential in the number of robots and objects, since it inherits the complexity of the underlying multi-robot motion planning problem. Nevertheless, we claim the hybrid problem representation and the heuristic can, in numerous realistic situations, guide the system such that its explores only a small part of the search space.

5.5.2 Applying a roadmap expansion action

Applying a roadmap expansion action is adding nodes[7] in some roadmaps. A roadmaps expansion action is linked to an applicable action. The target parts of roadmap are automatically computed by analyzing the preconditions and the effects of the correspondent applicable action (see §5.4.1 for an example). The insertion of nodes is shared probabilistically between all the roadmaps involved. It is also at this level that the planner can expand the Relative Roadmaps.

An important feature here, is that the insertion of nodes is done by taking into account the geometric context in which the roadmaps expansion action is achieved. This is to avoid to add nodes in collision with the other robots or objects at the moment where we would use them to find a path. More precisely, when we add new nodes, we select randomly one configuration belonging to cs_s of the previous state to become the collision context. As the search is forward, the previous state has at least one configuration belonging to cs_s. In almost all our problems, this feature improves drastically the efficiency of the planner.

Adding nodes during the search process is a key feature of our planner. It gives it the ability to adapt the addition of nodes to the real needs of the problem resolution.

5.6 Completeness and discussion

Completeness. In the worst case, all the symbolic plans can be explored. At the symbolic level, the planner is complete. The Validation algorithm is also complete. So if we work with precomputed roadmaps, the aSyMov planner is complete with the level of geometric planning knowledge available in the roadmaps. Consequently, if the planner does not find an existing solution, this is due to the fact that roadmaps are not sufficient.

The roadmaps methods are known to be complete in probability. If we take into account the expansion of the roadmaps during the search, aSyMov keeps this property. This is due to the fact that:

- The actions that are not yet validated have always a chance to be selected and validated
- When the roadmaps are expanded, the new nodes are associated to previously explored states.
- The roadmap expansion actions stay in the front search.

When the planner faces to solve problem that has no solution, it alternates indefinitely between adding nodes and trying to validate actions.

Search heuristic. The planner is first encouraged to find a plan close to the shortest in terms of number of actions. The farther a solution plan is from this first solution, the greater the effort will be to find it. We also use costs based on previous failures: the more an action fails to be validated, the less often it will be chosen. In this case the planner estimates that the validation of the action faces a geometric impossibility or a very constrained sub-problem. This last method provides an informative link between the symbolic representation and the geometric environment.

7. In the current implementation we have fixed arbitrarily the number of nodes to add at 5.

Limitation of the Configuration Space explored. One of the main points of the interweaving of the task planning process and the motion planning one is the limitation of the Configuration Space explored on the basis of the symbolic constraints. First, aSyMov explores only the parts of CS involved by the propositions belonging to $PLACE$ in the symbolic plans. If a proposition is not present in any symbolic plan, the corresponding set of configurations will not be considered. For example, if in a given problem, a robot is not useful for solving a problem, the transit and transfer roadmaps of this robot will not be expanded. Secondly, the symbolic constraints also give an order of exploration which limits the Configuration Space. For example, if the symbolic constraints indicate that a motion of a $robot\ 1$ must be achieved before a motion of a $robot\ 2$, the space where $robot\ 2$ moves first is not explored. Our planner is able to take advantage of strong symbolic constraints as discussed above. On the other hand, our planner has the same limitations as a motion planner when faced to in highly dimensional CS without symbolic constraints, even with the proposed two-step (multi-roadmaps, then validation) exploration algorithm.

5.7 Example

We give here below an a sketch of the overall procedure for the first steps of the *radio-switch* problem resolution. Let us assume that we have the pre-computed roadmaps (figure 4)[8]. The initial state is $s0 = (sym_{s0}, CS_{s0})$. CS_{s0} is linked to the configuration ($f0$, $p0$, $a0$) (see figure 10).

Let us imagine that the first action selected is:

- `GOTO_ARMBOT_INIT_SWITCH`

At the beginning of the resolution, this action seems to have the greatest interest. To validate this action, we need to find a node that is reachable from $a0$ and which belongs to the set of configurations where the end effector of $armbot$ touches the switch (this set was previously named $CS1$ in §3.2). The unique possible node is $a2$. Unfortunately, paths do not exist to reach this node in the geometric context of the initial state. $cbox$ blocks the path. The action is not validated. The planner has the choice now between:

- four roadmaps expansion actions:
  - `EXPAND_GOTO_FORKLIFT_INIT_CANGRASP_CBOX`
  - `EXPAND_GOTO_ARMBOT_INIT_SWITCH`
  - `EXPAND_GOTO_ARMBOT_INIT_ANYWHERE`
  - `EXPAND_GOTO_FORKLIFT_INIT_ANYWHERE`
- and three applicable actions:
  - `GOTO_FORKLIFT_INIT_CANGRASP_CBOX`
  - `GOTO_FORKLIFT_INIT_ANYWHERE`
  - `GOTO_ARMBOT_INIT_ANYWHERE`

8. aSyMov can be called with precomputed roadmaps

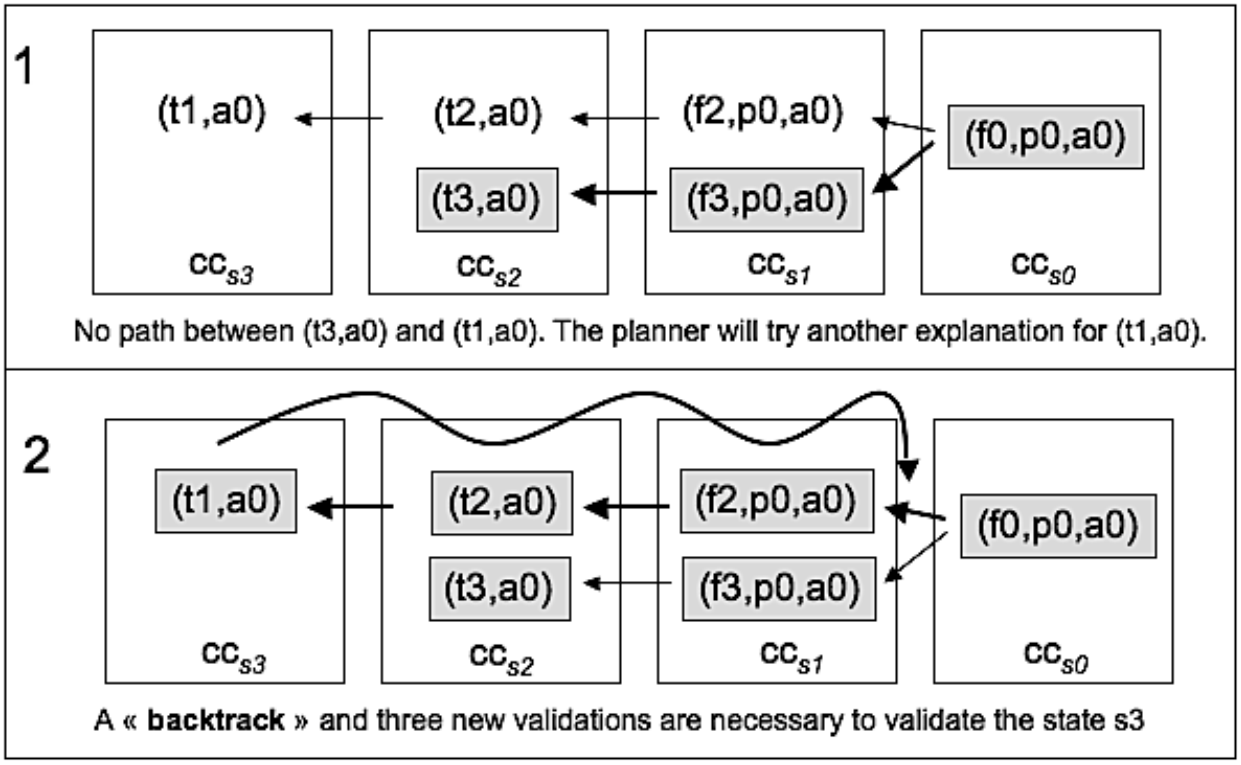

Figure 10: The set of candidate configurations during the resolution of the *radio-switch* problem

For the sake of simplicity, we suppose that the planner does not choose a *roadmaps expansion* action. It tries to validate the applicable action `GOTO_FORKLIFT_INIT_CANGRASP_CBOX`. The other actions do not have symbolic interest. In other words, they do produce a symbolic state which is closer to the symbolic goal. However, they might be used later to find intermediate configurations.

We have already shown in §4.2.1 that the computation of the set of candidate configuration for $s1$ gives $cc_{s1} = \{((f3),(p0),(a0)),((f2),(p0),(a0))\}$. Let us suppose that the validation algorithm has proven that $((f3),(p0),(a0)) \in cs_{s1}$. The state $s1$ is valid.

Then let us imagine that the planner remains on the same branch of the search space and now, it tries to validate the action:

```
(:action GRASP_FORKLIFT_CBOX_INIT
 :precondition
  P_FORKLIFT_CANGRASP_CBOX
P_CBOX_INIT
 :effect
  not P_CBOX_INIT
not P_FORKLIFT_CANGRASP_CBOX
  P_FORKLIFT.CBOX_CANUNGRASP)
```

Problem type	CPU (seconds)	success rate
1 forklift 1 box	12.87	100%
1 forklift 2 boxes	49.94	45%
2 forklifts 2 boxes	148.82	15%

Table 1: Average results on 20 tests for 3 kinds of "Forklifts and boxes" problems.

This action leads state $s2$; the set of candidate configuration is $cc_{s2} = \{((t3),(a0)),((t2),(a0))\}$. The validation algorithm, which is a lazy process, uses the last geometric instantiation to provide $((t3),(a0)) \in cs_{s2}$.

Still, if we imagine that the planner remains in the same branch of the search space, it will then try to validate:

```
(:action GOTO_FORKLIFT.CBOX_CANUNGRASP_GOOD_RECEPTION
 :precondition
  P_FORKLIFT.CBOX_CANUNGRASP
 :effect
  not P_FORKLIFT.CBOX_CANUNGRASP
  P_FORKLIFT.CBOX_GOOD_RECEPTION)
```

This action leads to $s3$. After computing the candidate configurations, the planner realizes there is no path between the previous validated configurations $((t3),(a0))$ and the only one candidate configuration $((t1),(a0))$ (see figure 10-1). Consequently, it tries to explore the other candidate configuration levels. Let us imagine that in the first action the second candidate can be validated, now $cs_{s1} = \{((f3),(p0),(a0)),((f2),(p0),(a0))\}$. We propagate it to the second action. Now $cs_{s2} = \{((t3),(a0)),((t2),(a0))\}$. Let us suppose that we now can reach $((t1),(a0))$ from $(t2,a0)$ without collision (see figure 10-2). The new state $s3$ is validated (figure 10-2)...

6. Empirical evaluation of the system

We have run the test 20 times with randomly generated problem instances of the *radio-switch* class on a 3.2 GHz Pentium 4 HT. All runs were successful, with an average computation time of 18.6 seconds.

In order to highlight the variety of problems that aSyMov can solve, we present below three other problem classes and the results obtained. We will also analyse the limitations of the current system.

6.1 Forklifts and boxes

In this problem class, forklifts must move boxes. We have tested various cases by changing the initial and final configurations and by choosing the number of robots and boxes involved. Figure 11 illustrates two of these problems.

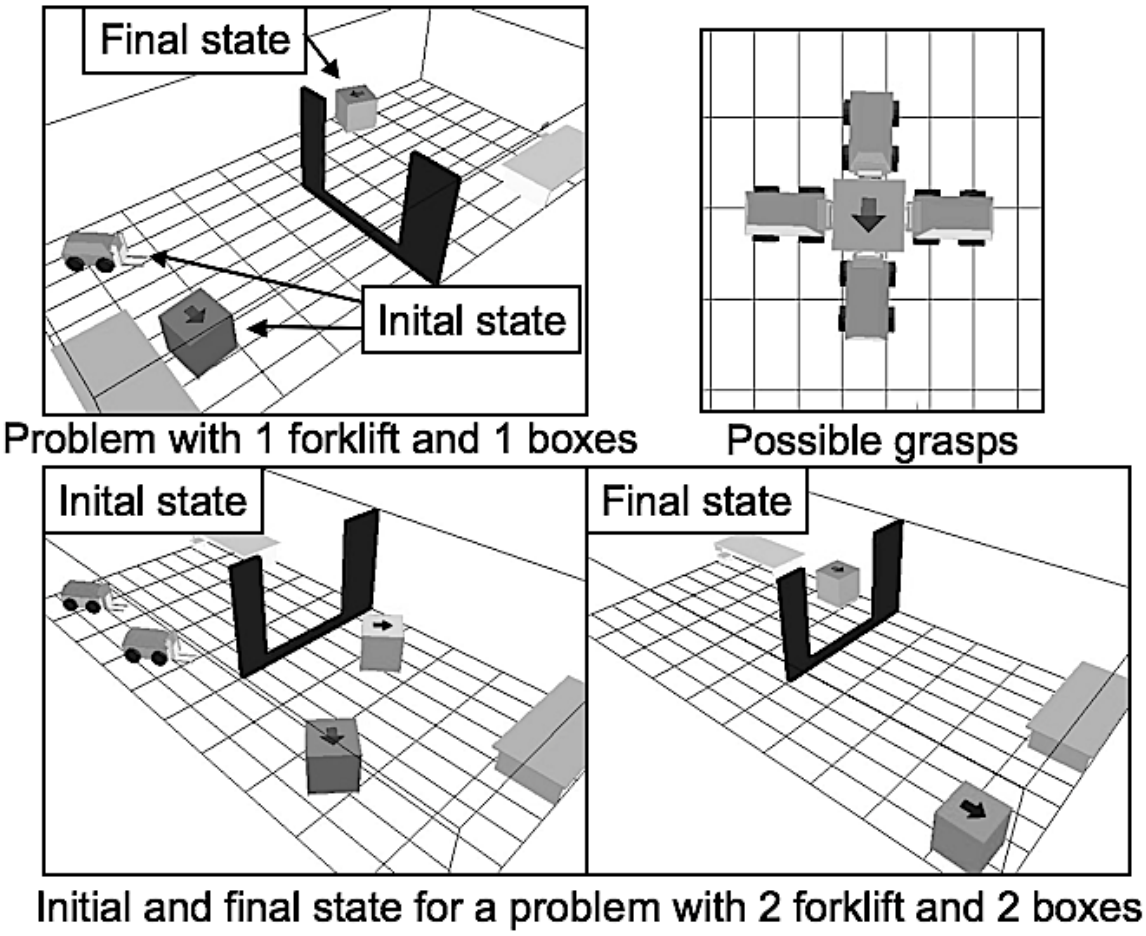

Figure 11: Examples of “forklifts and boxes” problems

Table 1 summarizes the obtained results. Theoretically, the success should be 100% for each type of problems if we give enough time to the planner. All computations have been given a maximum time of 500 seconds.

The number of geometric interactions grows exponentially with the number of robots or objects. This is not a surprise regarding the dimensions of the corresponding CS and the fact that the planner cannot take advantage of a symbolic constraints. It is in fact a pure geometric manipulation planning problem with a branching factor that increases when several robots can achieve the same tasks.

On the contrary to this extreme case, the following problems exhibit strong symbolic constraints and intricate links with geometry.

	success rate	CPU(seconds)	nb actions of the plan	nb actions optimal
1	100 %	9.97	28.8	28
2	78 %	78.2	16.3	?
3	100 %	298.9	40.16	28
4	100 %	271.6	44	44

Table 2: Average results on 50 runs for the 4 instances of the Geometric Hanoi Tower problem.

6.2 The Geometric Hanoi Tower problem

This class of problems is directly derived from the well-known task planning problem. Our planner can solve a three-cylinders problem respecting motion and manipulation constraints. We obtain different instances of this class of problems by tuning the dimensions of the central obstacle. One or two robots can be used to solve the problem. Figure 12 illustrates four problems of this class.

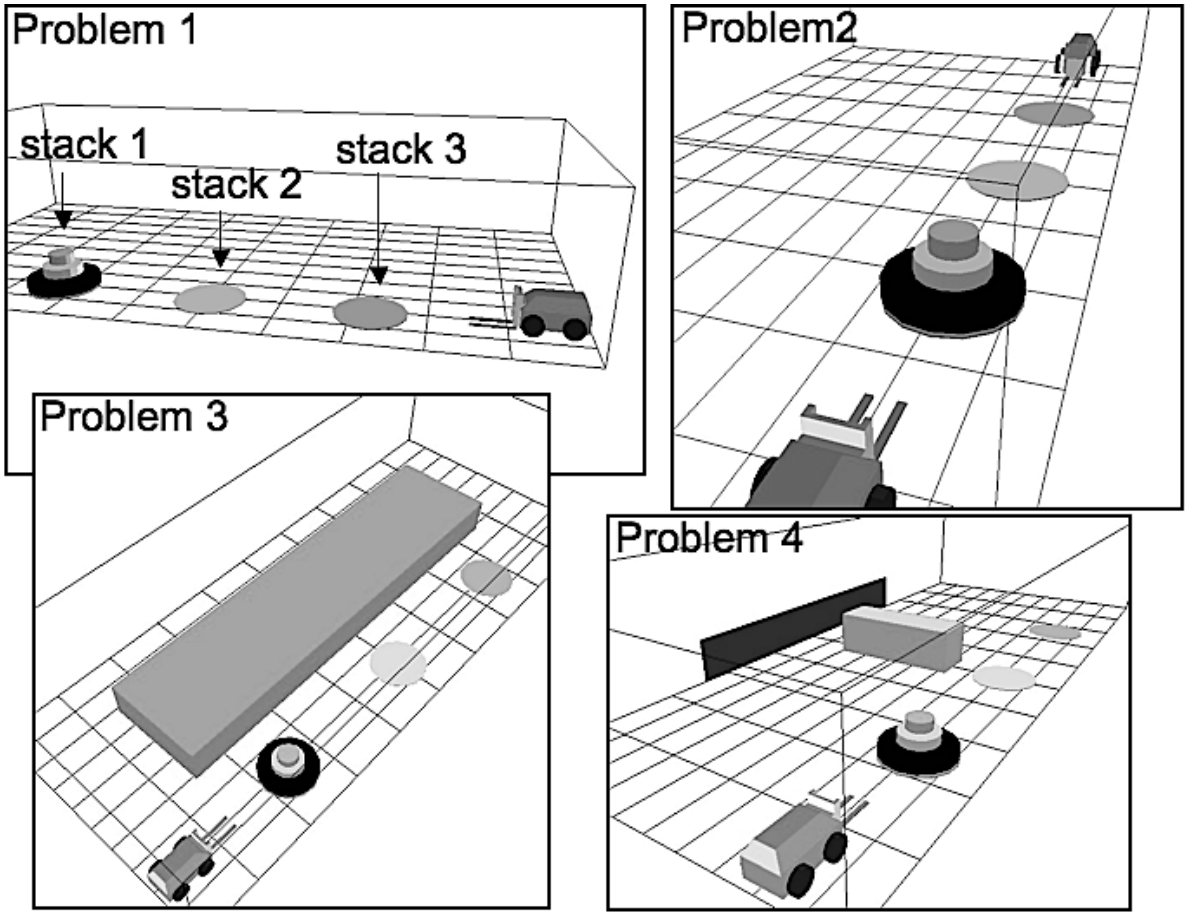

Figure 12: Four problems from the “Geometric” Hanoi Tower problem class

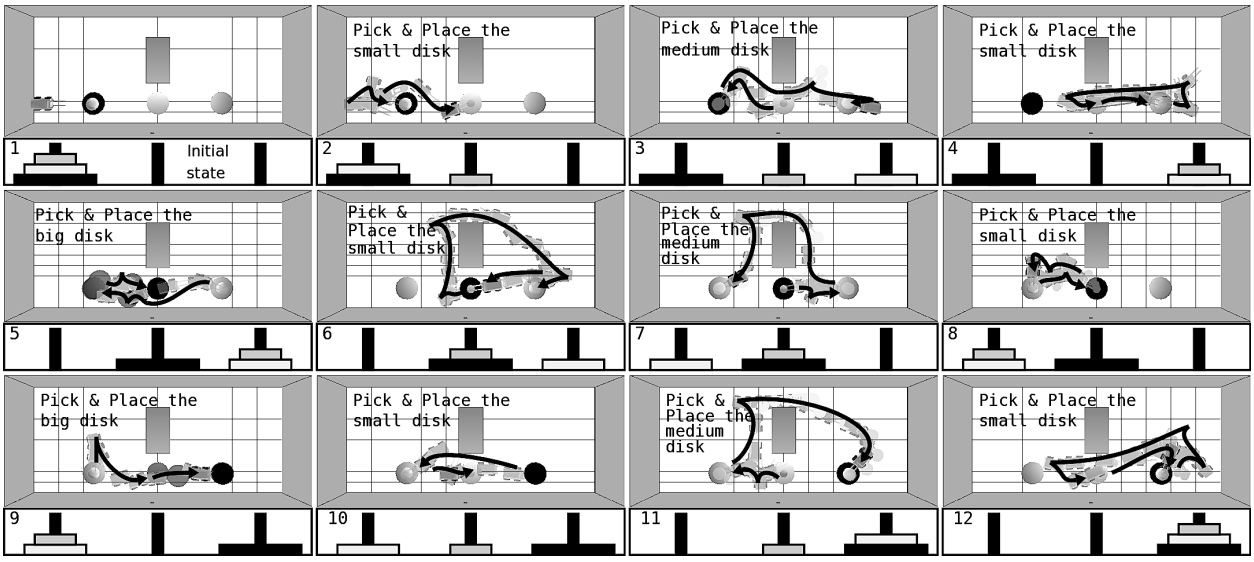

Figure 13: A plan found for Problem 4 of the Geometric Hanoi Tower class: actions and associated paths. This is the shortest possible plan (in terms of number of actions) because there is no possible path for the robot to transfer the Big Disk directly from `stack-1` to `stack-3` if a disk is placed on `stack-2`.

This class of problems permits to show how geometric constraints can influence qualitatively the overall solution. Indeed for Problem 4 (see figure 13), the high level plan is completely different from the plan that solves Problem 1, for which we obtain the classical solution where the big disk can be directly transferred from stack 1 to stack 3. In Problem 4, because of the geometric constraints, it is impossible to achieve this operation.

In Problem 3, even if the central obstacle is big, it does not create the same constraints as for Problem 4. We can find solutions close to those of Problem 1 or to those of Problem 4. This came as a surprise for us.

Note also that it is more difficult to find solutions for Problem 2 than for Problem 1. At task level, this problem can appear as easier: the second robot can be seen as a fourth stack. Unfortunately, there is a geometric complexity brought with the second robot as mentioned above. Note however that the planner solves the complete problem: its gives not only a symbolic plan but also the robot paths.

The Geometric Hanoi Tower problem allows to exhibit how the method tries to reduce the number of studied motions. Table 3 compares the number of studied motions in the validation process to the number of possible motions in the problem (counted “by hand”). Obviously, one motion is considered as different from another if the geometric contexts (i.e. the current configurations of the objects or the robots) are different.

Another interesting issue concerns the number of nodes developed in the roadmaps and how it can be limited and adapted to the effective intricacy of the situation when it is encountered. Indeed, the

	number of studied motions	number of needed motions for an optimal plan	number of possible motions
1	25.4	14	151
4	71.2	22	151
2	74.4	?	16148

Table 3: Number of studied motions in several instances of the Geometric Hanoi Tower problem class.

roadmap	transit	transfert Small Cylinder	transfert Medium Cylinder	transfert Big Cylinder
nb motions in the plan	7.04	4.02	2.16	1.0
nb of nodes	194.9	43.86	23.52	15.68

Table 4: Average number of nodes developed in the roadmaps compared to the number of motions needed for Problem 1 of the Geometric Hanoi Tower problem class.

roadmaps expansion is mainly controlled by the symbolic reasoning process. Table 4 illustrates the number of nodes created for each roadmap involved in Geometric Hanoi Tower Problem 1. The table presents average values computed on the basis of 50 runs. —-

6.3 An "extreme" case: the "IKEA" problem

Two robots, $Assembler$ and $Gluer$, must assemble a kit table. Only $Gluer$ can apply glue. Both robots can assemble and transfer objects. Two legs, a board and a glue dispenser are available. Before assembling a leg with the board, $Gluer$ must put two points of glue on the board. It has a glue tank with a capacity limited to two points of glue. It may refill itself at the glue dispenser. It starts with an empty glue tank. Another constraint: the board and the assembled table have to be transported by the two robots. The grasping configurations are continuously defined along the objects.

This problem is too complex to be fully solved by aSymov in its current implementation (it involves about 40 different roadmaps including roadmaps where both robot transport the table with various grasps). Here we have 2 robots and 3 objects and the two robots share almost the same skills, which augments the number of applicable actions. The use of continuous grasps is also an additional source of difficulty. However, the planner is able to find and instantiate plans using a mixed initiative procedure: the programmer gives a hint (best applicable action) at a some critical instant of the resolution. Thus, we essentially illustrate here the ability of aSymov to "digest" substantially complex problems.

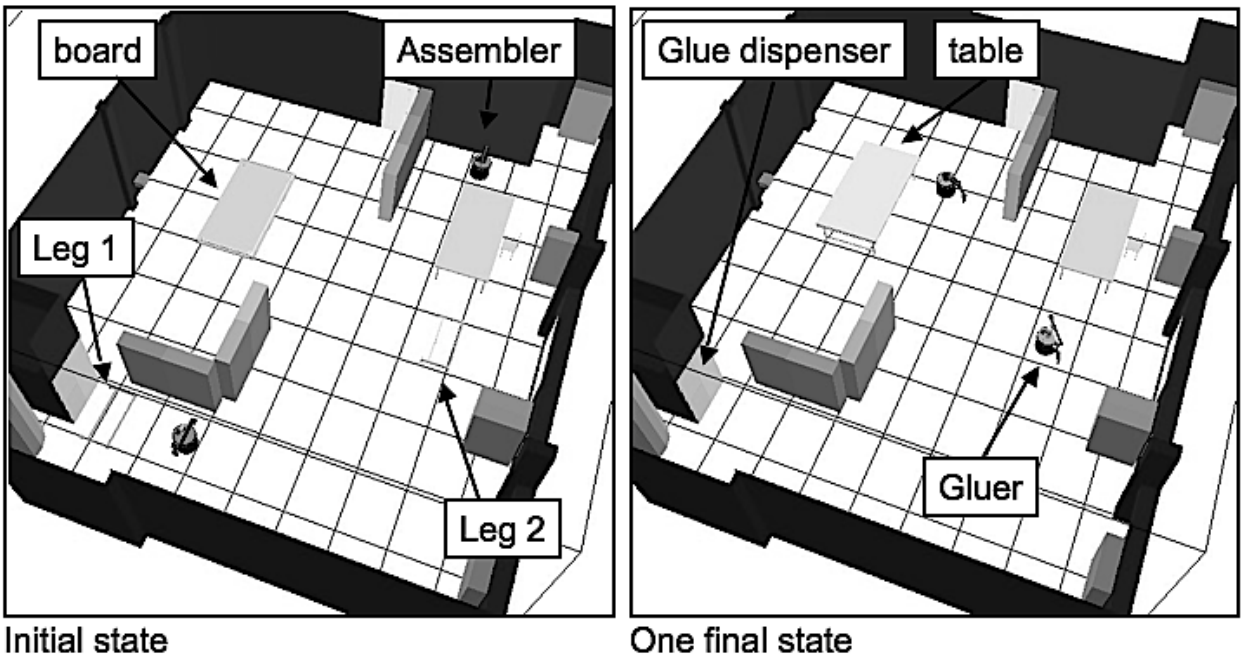

Figure 14: The “IKEA” problem.

6.4 Final comments

From the examples, we can confirm that the planner solves problems that have never been solved by other planners and that cannot be solved by a hierarchical system that simply uses the “best” task planner on top of the “best” motion planner.

We have also, voluntarily placed it in front of situation that it cannot solve completely in its current implementation in order to get some insights on the necessary improvements.

Two issues have clearly to be improved: the task planner heuristic search in order to take full advantage of the knowledge acquired incrementally through the roadmaps exploration, and the adaptation of more elaborate motion planning techniques.

The main reason why we used FF instance, was the fact that it is a very fast for classical task planning problems. The system would certainly take advantage of a more specific task planner that integrates the possibility to add new symbols (when new connected components appear in the roadmaps) or to merge symbols (when the roadmap expansion action finds a way to connect two components). Another key issue would be to use or implement cost based task planner algorithms that would be fed by cost estimations based on trajectory lengths.

Several improvements can be also be made at motion planning level. For instance, by integrating several techniques proposed in the literature for more oriented search in the C-space submanifolds, using techniques that provide more flexible roadmaps (Jaillet and Siméon (2006)) or that allow to efficiently synchronize multi-robot paths (Svestka and Overmars (1998); Qutub et al. (1998); Siméon et al. (2002); Clark et al. (2003); LaValle (2006)).

7. Conclusion

In this paper, we propose an original and systematic approach towards integrating symbolic task planning, and geometric motion and manipulation planning.

We introduced a new extension to classical action planning formalisms based on STRIPS-like description where manipulation and rearrangement planning problems in 3D environments are rigorously introduced and where reachability conditions must not be asserted by a programmer but are automatically inferred by checking for the existence of feasible paths using motion planning mechanisms. Our formalisms are independent from the exploration methods of the symbolic state space as well as from the exploration methods for the Configuration Space or the representation of the geometry (2D, 3D). Moreover, this formalism can express a pure task planning problem as well as a pure motion/manipulation planning problem.

On this formal basis, we have built a planner that is able to solve intricate geometric and symbolic constraints. We relied on an original exploration method of the Configuration Space which let us deal with complex problems of multi-robot manipulation planning. To cope with the complexity of the problems we have adopted a two-step method where we first takes into account the static geometric environment and then the interactions between the movable artifacts of the world.

The search process is guided by a symbolic level that solves a relaxed version of the problem in which all paths are considered as valid. The planner, at each step, tries to balance between (1) trying to find a plan with the level of knowledge it already has, or (2) "investing" more in a deeper knowledge of the topology of the different configuration spaces it manipulates.

We have evaluated the planner with very different instances of intricate problems. We illustrated, through experiments, the relevance of our approach which tends to limit the number of motions studied and the number of nodes in the roadmaps and to adapt dynamically to the intricacy of the situations. Nevertheless, the planner is more efficient when the problem is well constrained at the symbolic level. Indeed, it is able to exploit the symbolic constraints to reduce the exploration of high dimensional Configuration Spaces. Finally we have pointed out several potential improvements to the scheme based or more elaborate task and motion planning algorithms.

References

J.M. Ahuactzin, K. Gupta, and E. Mazer. Manipulation planning for redundant robots: a practical approach. In *International Journal of Robotics Research, Vol. 17, No. 7, 731-747*, 1998.

R. Alami, T. Siméon, and J.P. Laumond. A Geometrical approach to planning manipulation tasks. the case of Discrete placements and Grasps. In *H. Miura and S. Arimoto (Eds), Robotics Research : The Fifth International Symposium, pp 453-463, The MIT Press Massachusetts*, 1990.

R. Alami, J.-P. Laumond, and T.Siméon. Two manipulation planning algorithms. In *Algorithmic Foundations of Robotics (WAFR'94), A.K. Peters Pub., Boston, MA*, 1994.

R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4):315–337, 1998.

J. Barraquand and J.-C. Latombe. Robot Motion Planning: A Distributed Representation Approach. *International Journal of Robotics Research*, 10(6):628–649, 1991.

T. Bretl. Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. In *International Journal of Robotics Research, vol. 25: pp. 317 - 342.*, 2006.

S. Cambon, F. Gravot, and R. Alami. A robot task planner that merges symbolic and geometric reasonning. *16th European Conference on Artificial Intelligence (ECAI'2004), Valence (Spain), pp.895-899*, 2004.

T. Cao and A.C Sanderson. Task decomposition and analysis of robotic assembly task plans using petri nets. In *IEEE Transactions on Industrial Electronics, Volume: 41, Issue: 6*, 1994.

P.C. Chen and Y.K. Hwang. Practical path planning among movable obstacles. In *IEEE International Conference on Robotics and Automation*, 1991.

C.M. Clark, S.M. Rock, and J.C. Latombe. Dynamic networks for motion planning in multi-robot space systems. In *7th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2003.

G.E. Fainekos, Kress-Gazit, and G.J H. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *IEEE Conference on Decision and Control*, 2005.

R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving. In *2nd International Joint Conference on Artificial Intelligence (IJCAI-71), 608-620*, 1971.

E. Fink and M. Veloso. Formalizing the prodigy planning algorithm. In *New Directions in AI Planning, pp. 261-272, IOS Press*, 1996.

M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 2003.

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning theory and practice*. Elsevier, 2004.

F. Gravot and R. Alami. An extension for handling multiple roadmaps and its use for complex manipulation planning. In *Proc. IEEE International Conference on Robotics and Automation*, 2003.

F. Gravot, R. Alami, and T. Siméon. Playing with several roadmaps to solve manipulation problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems - IROS'02*, 2002.

F. Gravot, S. Cambon, and R. Alami. aSyMov: a planner that deals with intricate symbolic and geometric problems. *11th International Symposium on Robotics Research (ISRR'2003), Siena (Italy)*, 2003.

D. Halperin, J.-C. Latombe, and R.H. Wilson. A general framework for assembly planning: the motion space approach. In *14th symposium on Computational Geometry*, 1998.

J. Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20(20):291–341, 2003.

J. Hoffmann, S. Edelkamp, S. Thíebaux, R. Englert, F. Liporace, and S. Trüg. Engineering benchmarks for planning: the domains used in the deterministic part of ipc-4. *jair = "Journal of Artificial Intelligence Research"*, 2006.

S. Hutchinson and A. C. Kak. Spar: A planner that satisfies operational and geometric goals in uncertain environments. *AI Magazine*, 11(1):30–61, 1990.

L. Jaillet and T. Siméon. Path deformation roadmaps. In *7th Int. Workshop on Algorithmic Foundations of Robotics*, 2006.

L.E. Kavraki and P Svestka. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 1996.

Y. Koga and J.C. Latombe. On multi-arm manipulation planning. In *IEEE Int. Conf. on Robotics and Automation*, 1994.

J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Darien, CT, 1991.

C. Laugier and J. Troccaz. Sharp, a system for automatic programming of manipulation robots. *Robotics Research : The third International Symposium*, 1985.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.

S. M. LaValle. Rapidly-exploring random trees: a new tools for path planning. Technical Report TR98-11, Computer Science Dpt, Iowa State University, 1998.

S.M. LaValle, J. Yakey, and L.E. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *IEEE International Conference on Robotics and Automation*, 1999.

F. Lingelbach. Path planning for mobile manipulation using probabilistic cell decomposition. In *IEEE/RSJ Intelligent Robots and Systems*, 2004.

T. Lozano-Pérez. Spatial planning: A configuration space approach. In *IEEE Transaction on Computers Volume 32 , Issue 2, pages 108-120*, 1983.

T. Lozano-Perez, J.L. Jones, and E. Mazer. Handey: A robot system that recognizes, plans and manipulates. In *IEEE International Conference on Robotics and Automation, pages 843- 849*, 1987.

K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *International Journal of Robotics Research, vol. 15: pp. 533 - 556*, 1996.

I. Mazon, R. Alami, and P. Violero. Automatic planning of pick and place operations in presence of uncertainties. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'90*, 1990.

Ch. Nielsen and L. Kavraki. A two-level fuzzy prm for manipulation planning. In *IEEE Int. Conf. on Intelligent Robots and Systems*, 2000.

P.A. O'Donnell and T. Lozano-Perez. Deadlock-free and collision-free coordination of two robot manipulators. In *IEEE Robotics and Automation Conference 1989*, 1989.

E. Plaku, L. E. Kavraki, and M. Y. Vardi. A motion planner for a hybrid robotic system with kinodynamic constraints. pages 692–697, IEEE International Conference on Robotics and Automation 2007, Rome, Italy, 2007.

S. Qutub, R. Alami, and F. Ingrand. A scheme for coordinating multi-robot planning activities and plans execution. In *13th European Conference on Artificial Intelligence*, 1998.

P. Regnier and B. Fade. Complete determination of parallel actions and temporal optimization in linear plans of action. In *Proc. European Workshop on Planning*, 1991.

N. Rugg-Gunn and S. Cameron. A formal semantics for multiple vehicle task and motion planning. In *IEEE International Conference on Robotics and Automation, San Diego, CA, USA, Vol 3, pages 2564-2469*, 1994.

A.C. Sanderson, H. Zhang, and L.S. Homem de Mello. Assembly sequence planning. In *AI Magazine, Volume 11, Issue 1*, 1990.

J.P. Saut, A. Sahbani, S. El-Khoury, and V. Perdereau. Dexterous manipulation planning using probabilistic roadmaps in continuous grasp subspaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

T. Siméon, J.P.Laumond, and C.Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Journal of Advanced Robotics*, 14:477–494, 2000.

T. Siméon, J.-P. Laumond, and F. Lamiraux. Move3d: a generic platform for path planning. *4th International Symposium on Assembly and Task Planning*, 2001.

T. Siméon, S. Leroy, and J.-P. Laumond. Path coordination for multiple mobile robots: a resolution complete. In *IEEE Transaction on Robotics and Automation*, 2002.

T. Siméon, J.-P. Laumond, J. Cortès, and A. Sahabani. Manipulation planning with probabilistic roadmaps. *International Journal Robotic Research*, 20(20):291–341, 2003.

M. Stilman and J. Kuffner. Planning among movable obstacles with artificial constraints. In *Workshop on the Algorithmic Foundations of Robotics*, 2006.

M. Stilman and J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Proceedings of the 2004 IEEE International Conference on Humanoid Robotics (Humanoids'04)*, December 2004.

P. Svestka and M. Overmars. Coordinated path planning for multiple robots. In *Robotics and Autonomous Systems, 23 (1998), pp. 125-152.*, 1998.

C. P. Tung and A. C. Kak. Integrating sensing, task planning, and execution for robotic assembly. In *IEEE Transactions on Robotics and Automation*, 1996.

G. Wilfong. Motion planning in the presence of movable obstacles. In ACM *Symposium on Computational Geometry*, 1988.

E. Yoshida, M. Poirier, J.P. Laumond, R. Alami, and K.Yokoi. Pivoting based manipulation by humanoids: a controllability analysis. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.