Let's reduce the gap between task planning and motion planning

Emmanuel Guere, Rachid Alami

Let's reduce the gap between task planning and motion planning

Emmanuel Guéré, Rachid Alami

LAAS-CNRS
7, avenue du Colonel-Roche 31077 Toulouse Cedex - France
{guere,rachid}@laas.fr

Abstract

In the stream of research that aims to speed up practical planners, we propose a new approach to task planning based on Probabilistic Roadmap Methods (PRM). Our contribution is twofold. The first issue concerns the development of ShaPer, a task planner that is able to deal efficiently with large problems. Shaper "captures" the structure of the task space. The second contribution involves promising results on robot task planning. This is obtained through an analysis of the task space structure that exhibits the relation between task and geometric reasoning for a given robot task. To illustrate such an approach, we solve a complex problem where motion and task planning are closely interleaved.

1 Motivations

Task planning has often used examples borrowed from robotics like, for instance, Pick&Place scenarii. However, the effective use of practical task planners in robotics has always been limited to domains where it was possible to establish a clear and impermeable hierarchy between a high-level task planner and a lower level where geometric problems are dealt with. This is clearly insufficient if one wants to tackle realistic robotics problems. For instance, a plan for building a stack of objects may be substantially modified if one adds an obstacle or changes the shape of the robot[10, 8].

We have proposed in the early nineties a geometrical formulation of the manipulation problem[2]. We formulated the problem as a series of motion planning problems in presence of movable obstacles. We showed that it was possible to compute regions in the global configuration of the system where a grasp or a release action may cause a qualitative change in the topology of the free space allowing to access new states in the task space of a given manipulation problem. Such regions correspond to links between various "slices" of the global configuration space[1].

While the formulation was satisfactory and gave a deep understanding of the manipulation problem, its effective application has been limited to environments with a small number of degrees of freedom[1].

Indeed, we faced three problems. The first one was due to the limitations of the motion planners of that (old) times. It was unrealistic to try to solve motion problems with more than 3 degrees of freedom. The second problem was the absence of an operational link between task planning and motion planning. The third problem which was also discouraging was the slowness of task planners.

We are convinced that the recent and independent advances in motion and task planning have reached a level where it becomes realistic and fruitful to investigate the links between them and to devise paradigms that effectively involve the two aspects in close relation and not simply through a gross and somewhat artificial hierarchical decomposition. This paper is a first step toward this goal.

Even though task planners have made very substantial progress [3] over the last years, they are still limited in their use. There are also domains, like in robotics which heavily influence the structure of the task space; learning such a structure will certainly help in building an efficient planner in a given domain. However, the structure of the environment (at least the "useful" one) heavily depends not only on the environment but also on the actions that can be performed. Our aim is to develop a generic planner that will exhibit and learn the "structure" of a given domain. This is the reason why we propose to investigate approaches based on Probabilistic Roadmap (PRM). PRM basically "captures" the space "topology" through random configuration generation and connectivity tests between states using a local planner. PRM obtains good results in robot path planning because it is relatively easy to test the validity of a

randomly generated configuration and because there exist good metrics and numerous very efficient local planners. PRM can even obtain excellent results when careful techniques are devised in order to construct a compact graph and to "direct" the search toward non-explored regions[9].

Our contribution is twofold. The first issue concerns ShaPer[1], a task planner that is able to deal efficiently with large problems. We will show that, even if problems are modeled in first-order logic, there exists a topology for task planning domains. With a PRM like algorithm, we show how we can devise a task planner which builds a graph of the task space.

The second contribution involves promising preliminary results on joining task and geometrical reasoning. Indeed the environment definition and the robot configurations can be modeled in first-order logic. ShaPer's expressiveness coupling with a PRM like algorithm is also able to deal with geometrical constraints.

Both contributions are illustrated through a prototype implementation presented on a problem where motion and task planning are closely interleaved.

2 ShaPer: An efficient task planner

In this section, we briefly present ShaPer, a new version of the task planner proposed in [6]. Shaper is based on STRIPS formalism [4][2]. It performs in two steps: first an *accessibility* graph is generated *off-line*, and then the planner solves *online* task planning problems by extracting a plan from the learned graph. Even though usual task planners are limited by a combinatorial explosion, ShaPer is able to learn the *accessibility* graph for substantially large domains; this is possible because this graph only contains "relevant" states with respect to the state space "tolopology".

2.1 Relevant states

A new state is declared "relevant" if it allows to access to an unknown state (i.e. there is no other state in the current graph $\mathcal{G}$ with the same *shape*). We say that two states S and g have the same *shape* iff there exists a substitution σ such that $S = \sigma(g)$. Consider for instance the two block-world states g and S:

[1]ShaPer: Shape based Planner.

[2]An action o consists of three parts: P_o represents the preconditions, A_o the add-list (new facts - $A_o \cap P_o = \phi$) and D_o the Del-list (facts: not true when o is applied - $D_o \subseteq P_o$) represent the effects. o is applicable to a state S if and only if $P_o \subseteq S$. In this case $o(S) = (S - D_o) \cup A_o$.

Relevant(S', $\mathcal{G}$, Max_Trials): $Boolean$
  ForEach $g \in \mathcal{G}$
    If $\exists \sigma_{g,S'}$ **Then**
      Return false /*S' $shape$ already in $\mathcal{G}$ */
  $trial \leftarrow 0$
  $found \leftarrow true$
  While $trial < Max\_Trials$ **Do**
    Randomly choose Γ a local plan /*$P_\Gamma \subseteq S'$*/
    ForEach $S'' \in \mathcal{G}$ **Do**
      If $\mathcal{L}(S'', \Gamma(S'))$ **Then**
        $found \leftarrow false$ /*Already in $\mathcal{G}$*/
        $trial \leftarrow trial + 1$
        Break
    If $found$ **Then**
      Return true /*S' adds information*/
  Return false

Table 1: Relevance of a state.

with $g = \{Clear(A), On(A,B), OnTable(B), Clear(C), OnTable(C)\}$ and $S = \{Clear(A), OnTable(B), Clear(B), OnTable(C), On(A,C)\}$. The state g, when A is substituted by A, B by C and C by B is equal to S ($\sigma = \{A/A, C/B, B/C\}$). Thus, if P is a sequence of actions applicable to g, then $\sigma(P)$ is applicable to S and $(\sigma(P))(S) = \sigma(P(g))$. Such a graph $\mathcal{G}$ also contains all accessible shapes (of a given connexity). Table 1 presents an algorithm to test the relevance of a state.

2.2 An accessibility graph learning

In motion planning, PRM generates a random configuration, checks its validity (if it is in $\mathcal{C}_{free}$) and tries to connect it to the current graph. In task planning, there is no general way to check the validity of a state. This is the reason why we do not generate states randomly. Instead of that, we generate random valid plans starting from a given initial valid state (see table 2 [3]).

[3]We develop a graph with only one connected component including the initial state S_{begin} by applying valid local plans.

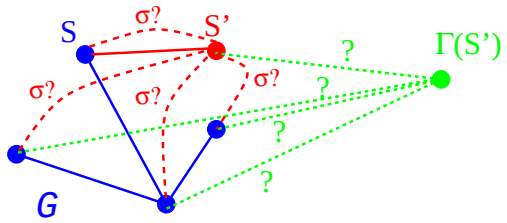

Figure 1: Is S' a relevant state for the graph $\mathcal{G}$?

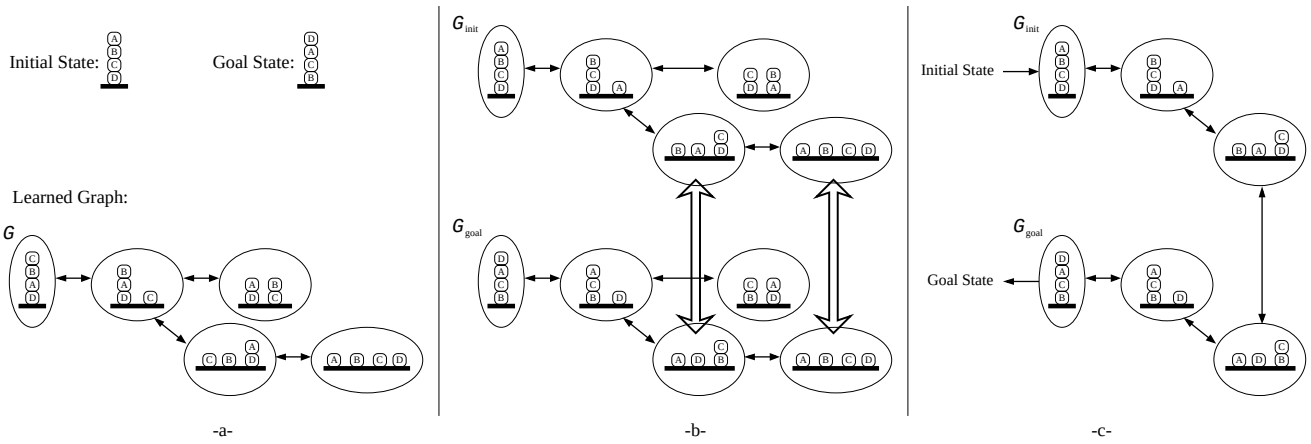

Figure 2: Solution extraction in block-world domain. a. The graph is generated *off-line* ; Initial and goal states are defined *online*. b. Create $\mathcal{G}_{init}$ and $\mathcal{G}_{goal}$ from the substitutions of S_{init} and S_{goal} in $\mathcal{G}$; connect $\mathcal{G}_{init}$ to $\mathcal{G}_{goal}$. c. Search a path in the graph.

After each random generation of a state S' by applying a random valid plan Γ, we must check the relevance of the new node $S' = \Gamma(S)$. To do so, we first look for a substitution σ between S' and $\mathcal{G}$. If it is not the case, S' is potentially relevant; we must then verify that S' allows to access to to a new state S'' that is not directly accessible from the current $\mathcal{G}$ (i.e. $\forall g \in \mathcal{G}.\neg\mathcal{L}(S'', g)$ - see Table 1 and figure 1).

Owing to the accessibility property of shapes (i.e. if $S = \sigma(g)$ and P a plan such that $S' = P(S)$ then $\sigma(S') = (\sigma(P))(g))$, even if the graph is limited to one substitution, we are able to extract sound plans.

2.3 Solution extraction

Note that the planner can not directly use the graph $\mathcal{G}$ to extract a solution. Indeed, the initial state S_{init}, the goal state S_{goal} and the graph $\mathcal{G}$ may correspond to "different substitutions" (see, for instance, figure 2.a).

So, the solution extraction[4] consists in four steps: 1. Build the graphs $\mathcal{G}_{init}$ and $\mathcal{G}_{goal}$ using S_{init} and S_{goal} substitutions; 2. Connect[5] $\mathcal{G}_{init}$ and $\mathcal{G}_{goal}$ with a local planner; 3. Connect S_{init} to $\mathcal{G}_{init}$ (S_i) and S_{goal} to $\mathcal{G}_{goal}$ (S_g); 4. Search for a path between S_i and S_g.

This new algorithm answers the problem of randomly generating a valid state in the method presented in [6].

[4]Note that two states S_1 and S_2 with the same *shape* are not necessary in the same connexity (i.e. there exists no path between $\mathcal{G}_1$ and $\mathcal{G}_2$). In such case, we can say that there exists no solutions.

[5]Note that although connection is always possible (if a plan exists), in restricted domains, to connect $\mathcal{G}_{init}$ and $\mathcal{G}_{goal}$, the planner may need to find intermediate substitutions.

2.4 Results for classical planning domains

Table 3 presents some problem resolutions for the block-world and the gripper domain. We compare ShaPer to IPP-v4.0 [7] one of the fastest task planners. The problems that have been used in the block-world domain are defined by: the initial state corresponds to the highest possible tower and the goal state to the same tower except for the top block that is put at the bottom. In the gripper domain, the problem consists in moving all balls from a table to another with a robot which can pick two balls at a time.

3 Predicate independence

While the previous section presented the general framework of our planner, there are still efforts to be dedicated in order to reduce the graph size. For instance, one can take advantage of predicate independence. Indeed, given two independent predicates f and h, and F (resp. H) the accessibility graphs constructed from f (resp. h), we would have, with the algorithm described above, $Card(\mathcal{G}) = Card(F) \cdot Card(H)$, whereas the construction of $Card(\mathcal{G}) = Card(F) + Card(H)$ would have been sufficient.

3.1 When?

Two predicates f and h are said to be independent, in a given domain, iff $\forall o \in \mathcal{O}.f \in A_o \cup D_o \rightarrow h \notin A_o \cup D_o$. Similarly, we define classes of independent operators: two operators o_1 and o_2 are independent iff $(D_{o_1} \cup A_{o_1}) \cap (D_{o_2} \cup A_{o_2}) = \phi$ Note that even if o_1 and

o_2 are independent, we can have $P_{o_1} \cap (A_{o_2} \cup D_{o_2}) \neq \phi$. This means that o_1 may need preconditions from other predicate classes. For instance, in a domain where the robot moves blocks from a table to another, *block-position* is independent of *robot-position*, but to pick a block (operator of the *block-position* class) the robot needs to be *near* the table.

3.2 How?

The algorithm presented in table 4 builds a more compact *accessibility* graph by taking into account predicate independence.

To better understand this algorithm, we run[6] it on a very simple domain. The environment is composed of three locations (L_1,L_2 and L_3 with $Table(L_1)$ and $Table(L_3)$), two blocks (B_1 and B_2) and a robot which can *move*, *pick* or *place* a block (the *Pick & Place* actions are symmetric):

Move(X:Location, Y:Location)
Pre: At(X), Connect(X,Y)
Add: At(Y)
Del: At(X)

Pick(X:Block, Y:Location)
Pre: At(Y), On(X,Y), HandEmpty, Table(Y)
Add: Hold(X)
Del: On(X,Y), HandEmpty

After predicate decomposition, we obtain two classes: $E_1 = \{On, HandEmpty, Hold\}$ and $E_2 = \{At\}$[7].

Let us first decompose the initial state into E_1 and E_2: $e_1^1 = \{On(B_1, L_1), On(B_2, L_1), HandEmpty\}_{E_1}$, $e_2^1 = \{At(L_1)\}_{E_2}$ and $c = \{Table(L_1), Table(L_3), Connect(L_1, L_2), Connect(L_2, L_1), Connect(L_2, L_3), Connect(L_3, L_2)\}_{Const.}$ Now we apply the algorithm step by step (see figure 3 for the state description):

[6] To simplify explanations, we present a deterministic version of table 4. The deterministic version develops all possible actions in all possible classes.

[7] Note that the predicates $Table$ and $Connect$ are not in any classes because no operator modifies them.

Learn_Graph(S_{init}, $Coverage$)
  $\mathcal{G} \leftarrow \{S_{init}\}$
  $cover \leftarrow 0$
  While $cover < Coverage$ **Do**
    Randomly choose $S \in \mathcal{G}$
    Randomly choose Γ a valid local plan
    Given $S' = \Gamma(S)$ /*$P_\Gamma \subseteq S$ */
    If Relevant(S',$\mathcal{G}$,Max_Trials) **Then**
      $\mathcal{G} \leftarrow \mathcal{G} \cup \{S'\}$
      $cover \leftarrow 0$
    Else $cover \leftarrow cover + 1$

Table 2: Learn the shape graph.

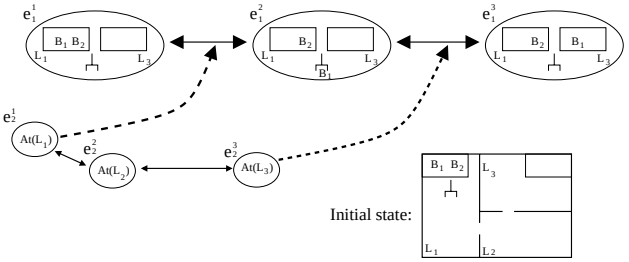

Figure 3: Algorithm with independent predicates explanation (dashed arrows represent preconditions).

1. No action is applicable to $e_1^1 \cup c$.
2. Apply $Pick(B_1, L_1)$ to $e_1^1 \cup e_2^1 \cup c$: create e_1^2 with precondition e_2^1.
3. Don't apply $Pick(B_2, L_1)$ to $e_1^1 \cup e_2^1 \cup c$ because of the substitution with e_1^2.
4. Apply $Move(L_1, L_2)$ to $e_2^1 \cup c$: create e_2^2 .
5. No action is applicable to $e_1^2 \cup c$.
6. Don't apply $Place(B_1, L_1)$ to $e_1^2 \cup e_2^1 \cup c$ because of the substitution with e_1^1.
7. Apply $Move(L_2, L_3)$ to $e_2^2 \cup c$: create e_2^3.
8. No action is applicable to $e_1^2 \cup c$.
9. Apply $Place(B_1, L_3)$ to $e_1^2 \cup e_2^3 \cup c$: create e_1^3 with precondition e_2^3.
10. No $Move$ applicable because of substitutions.
11. No action is applicable to $e_1^3 \cup c$.
12. Don't apply $Pick(B_2, L_1)$ to $e_1^3 \cup e_2^1 \cup c$ because of the substitution with e_1^2.
13. Don't apply $Pick(B_1, L_3)$ to $e_1^3 \cup e_2^3 \cup c$ because of the substitution with e_1^3.
14. No more action is applicable. The graph is complete.

The graph built with independent predicates (Table 4) is equivalent to the first one (Table 2). Consequently the solution extraction can be made as de-

Problem	IPP	ShaPer		
		Graph learning		Extraction
	CPU	nb_{node}	CPU	CPU
12 blocks	171.9	77	1.58	0.01
15 blocks	-	176	16.73	0.06
20 blocks	-	627	173.79	0.51
10 balls	56.9	30	0.07	0.01
20 balls	-	60	0.55	0.01
50 balls	-	150	14.04	0.15

Table 3: Graph learning and solution extraction for some classical domains

Learn_Graph(S_{init}, $Coverage$)
  $cover \leftarrow 0$
  $E_1 \times \ldots \times E_n \leftarrow Decompose\_Predicates(S_{init})$
  While $cover < Coverage$ **Do**
    ForEach E_i **Do**
      Randomly choose $e \in E_i$
      Randomly choose $o \in \mathcal{O}_i$ with $P_o \subseteq e$
      If $\neg\exists\sigma$ **Then**
        Add $(e - D_o) \cup A_o$ to E_i
        $cover \leftarrow 0$
      Else $cover \leftarrow cover + 1$
      ForEach e in completion E of E_i **Do**
        Randomly choose $o \in \mathcal{O}_i$ with $P_o \subseteq e$
        If $\neg\exists\sigma$ **Then**
          Add $(e - D_o) \cup A_o$ to E_i with precondition on E
          $cover \leftarrow 0$
        Else $cover \leftarrow cover + 1$

Table 4: Learning the shape graph with predicate independence.

scribed previously (with the independence transformation).

4 A step toward geometric reasoning

Now we show how we use the *accessibility* graph learning algorithm to integrate abstract reasoning (task planning) and geometrical constraints (motion planning). This is illustrated through an example.

4.1 Introduction of motion planning

In order to model numerical facts (e.g. Cartesian coordinates of the robot), we extend STRIPS formalism as following:

- numerical facts: $At(7.5, 1.3)$ models the robot position in Cartesian coordinates and $Size(Tank, 2.0)$ models the fact that the robot width is 2.0 when it holds the Tank;
- intersection and inequality in preconditions.

Move(X_1: **Real**,Y_1: **Real**,X_2: **Real**,Y_2: **Real**, S: **Real**)
  Pre: At(X_1, Y_1), Robot_Size(S),
  $\forall d \in \mathcal{E}_{nv}.d \cap Disc([(X_1, Y_1), (X_2, Y_2)], S) = \phi$
  Add: At(X_2, Y_2)
  Del: At(X_1, Y_1)

Pick(X:**Real**, Y:**Real**, Z:Block, T:Table, S:**Real** , S':**Real**)
  Pre: At(X,Y), HandEmpty, On(Z,T), Size(Z,S),
  Robot_Size(S'), Dist((X,Y),T) $\leq Arm_{lenght}$
  Add: Hold(Z), Robot_Size(S)
  Del: On(Z,T), HandEmpty , Robot_Size(S')

Owing to the interaction between predicates of the different classes (e.g. *On* and *At*), Shaper is able to build a graph for the robot position (and accessibility *via* motion) that depends on the robot width, instead of generating one graph per robot width. In this case, edges are labeled by the *Robot_Size* predicate (see the previous section).

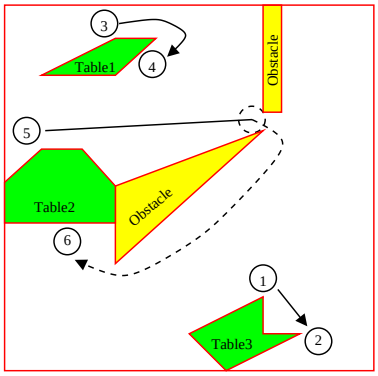

Figure 4: Environment description and "Grasp" classes explanations

Figure 5 presents a learned graph for the tank of water example (a 2D representation of the environment is presented in figure 4). The goal is to have a glass of water on $Table_3$ (initial position of the robot). To do this, the robot must move a tank of water (initially on $Table_1$) and a glass (initially there is one on $Table_1$ and another one on $Table_2$) to $Table_3$ where a pump allows to fill the glass from the tank.

What is of interest here is the capability of this scheme to provide an effective way to deal with intricate links between the logics of the task and its geometric counterpart.

First, because there is a narrow passage which prevents the robot to go from one side to the other while holding a big object (the tank), ShaPer maintains two classes of robot positions for picking objects on $Table_2$. This fact has drastic consequences on robot plans that need to transfer the tank. This is the reason why a STRIPS plan fails (i.e. it deals with a too high level environment model).

Figure 5 compares a plan obtained by using a classical two-level task planner and motion planner with a plan produced by ShaPer. Even if one interleaves a task planner and a motion planner (to avoid failure during execution), ShaPer's expressiveness allows to find the shortest plan (e.g. pick the *Glass* from $Table_2$ instead of $Table_1$).

4.2 Object grasping

In the geometric context, we must define the fact that the robot is able to *pick* or *place* an object. Indeed, *Move* action does not allow to know the table

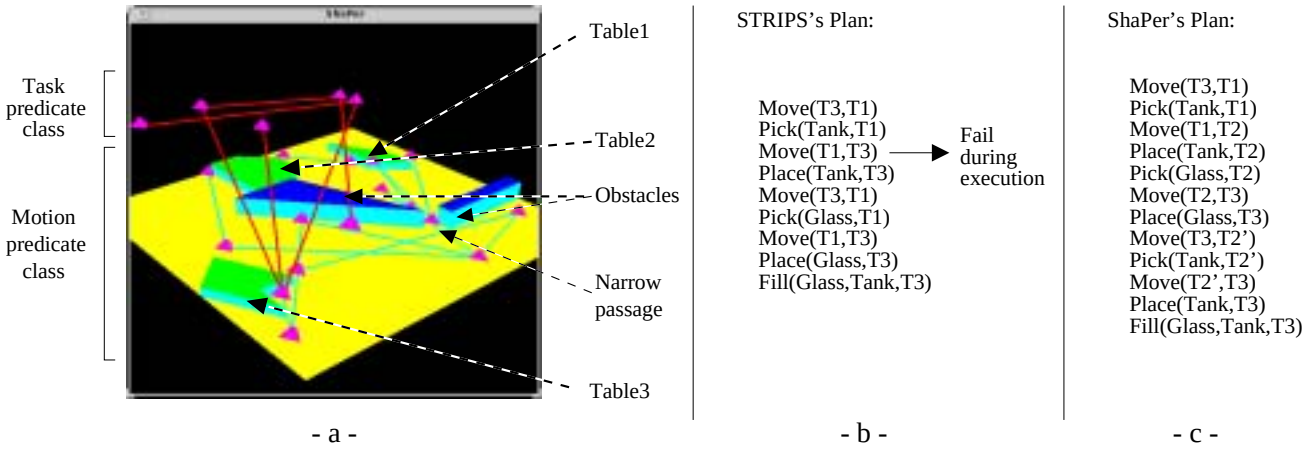

Figure 5: Tank of water problem. a. Learned graph with predicate independence. b. Plan in classical STRIPS model. c. Plan when the planner takes into account geometrical constraints.

proximity. A *near* precondition can, for instance, be defined based on robot-table distance.

Such a method allows infinite grasp positions; ShaPer allows to group them into several "grasp" classes (if necessary). Note that their number depends on the local planner ($\mathcal{L}$) capabilities. Figure 4 shows two distinct cases: i) $\mathcal{L}$ is a straight-line. The node n_2 is locally accessible from n_1, so n_2 is in the same class as n_1, it is not added. In the same way n_3 and n_4 define two classes; n_5 and n_6 define two classes. ii) $\mathcal{L}$ is a more powerful motion planner. n_4 is accessible from n_3, so there is only one class. However n_5 and n_6 define two classes because the robot width does not allow to pass through the narrow passage. Consequently the two nodes are necessary to capture the task topology.

5 Conclusion and future work

We have proposed a new planning algorithm based on an accessibility graph learning. ShaPer allows to demonstrate promising capabilities of such a method. Indeed, it is able to deal efficiently with complex task problems and geometric constraints. An example, where task and motion planning are closely interleaved, shows that ShaPer is more expressive than a hierarchical decomposition in a high level where task planning is performed and a lower level where geometric problems are dealt with.

Our future work will concern further investigations on the following aspects: i) improvement of the reasoning on robot manipulation [1] and ii) extension to deal with uncertainty especially by including perception actions [5].

References

[1] R. Alami, J.P. Laumond, and T. Siméon. Two manipulation planning algorithms. In *The First Workshop on the Algorithmic Foundations of Robotics, A.K. Peters Pub., Boston, MA.*, 1994.

[2] R. Alami, T. Siméon, and J.P. Laumond. A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps. In *In Robotics Research: The Fifth International Symposium, Tokyo (Japan)*, pages 113–123, 1990.

[3] A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, pages 281–300, 1997.

[4] R.E. Fikes and N.L. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[5] E. Guéré and R. Alami. A possibilistic planner that deals with non-determinism and contingency. *Proc. 16th Inter. Joint Conf. on Artificial Intelligence (IJCAI'99)*, 1999.

[6] E. Guéré and R. Alami. An accessibility graph learning approach for task planning in large domains. *ECAI 14th Workshop on Planning and Configuration*, 2000.

[7] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an adl subset. In *4th European Conference on Planning (ECP'97)*, 1997.

[8] J.-C. Latombe. Robot motion planning. In *Kluwer Press*, 1991.

[9] T. Simeon, J.P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *(Submitted to Advanced Robotics Journal) A short version appeared in IEEE IROS*, 1999.

[10] P. Tournassoud, T. Lozano-Perez, and E. Mazer. Regrasping. In *IEEE, International Conference on Robotics and Automation, Raleigh*, 1987.