# An accessibility graph learning approach for task planning in large domains

Emmanuel Guere, Rachid Alami

# An accessibility graph learning approach for task planning in large domains

Emmanuel Guéré and Rachid Alami

LAAS-CNRS,
7 avenue du colonel Roche,
31077 Toulouse, 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 an extension of GraphPlan[1] specially designed to deal with "local planning" in large domains. Having a reasonably efficient "local planner", we show how we can build a "global task planner" based on PRM and we discuss its advantages and limitations. The second contribution involves some preliminary results that allow to exploit to domain symmetries and to reduce in drastic manner the size of the "topological" graph. The approach is illustrated by a set of implemented examples that exhibit significant gains.

## 1 Introduction

Even though task planners have made very substantial progress over the last years, they are still limited in their use. This is the case with large domains where numerous facts and a huge number of possible actions instantiations are not relevant - a posteriori - for solving a given problem.

There are also domains, like in robotics, where the environment has a given topology; 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 state 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 in the configuration space. 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[12].

We propose an extension of these notions to task planning. Our contribution is twofold. The first issue concerns an extension of GraphPlan[1] specially designed to deal with "local planning" in large domains. Having an reasonably efficient "local planner", we show how we can build a first "global task planner" based on PRM and which builds a "topological" graph approximation of the task space.

We also discuss its advantages and limitations. The second contribution involves some preliminary results that allow to exploit domain symmetries and to reduce in a drastic manner the size of the "topological" graph. Both contributions are illustrated through a prototype implementation. The results are very promising.

## 2 Probabilistic Roadmap Method (PRM) background

### 2.1 Learning and Using

PRM [9] have been successfully used in path planning. A PRM planner performs in two steps: i) topology learning and ii) using the learned topology to search a solution of a given problem.

PRM builds a graph, $\mathcal{G} = (V, E)$, which "captures" the configuration space topology. The vertices $V$ correspond to randomly generated configurations, and the edges $E$ to the possible connections between vertices. A local planner $\mathcal{L}$ is used to test such a connection. Table 1 shows a basic version of PRM algorithms. The predicate $connect(v, q)$ means that configurations $q$ and $v$ are already connected by the graph. This test allows PRM to avoid cycles; indeed, $\mathcal{G}$ is limited to a tree in order to allow a significantly faster solution search.

To illustrate PRM algorithm, we develop a toy example in figure 1 with $MAX = 5$: PRM chooses randomly the configurations $c_1$ and $c_2$. In our example, the local planner $\mathcal{L}$ simply tests the existence of a collision-free straight line between two configurations. $\mathcal{L}$ can not find a path between $c_1$ and $c_2$. A new configuration $c_3$ is randomly generated; PRM creates a connection $a$ between $c_1$ and $c_3$ because of $\mathcal{L}(c_1, c_3)$. Adding $c_4$ creates a new connection $b$ with $c_3$. Then, $c_5$ allows to connect $c_1$ $(c)$ and $c_2$ $(d)$. When the learning step is stopped, one can use the graph to search for a global solution to a path planning problem. The initial $S$ and goal $S'$ states are first connected to the graph $G$ with the local planner. A search is then performed and obtains a collision-free path $(S \rightarrow c_3 \rightarrow c_1 \rightarrow c_5 \rightarrow c_2 \rightarrow S')$[1].

### 2.2 A visibility based algorithm

There is clearly a need to limit the size of the graph while maintaining the best possible "coverage". To do so, Move3D[12] proposes a PRM that computes "visibility" roadmaps which consist of two classes of nodes: the *guards* and the *connectors*. When a new valid configuration is randomly found, three cases may arise:

- either it is not visible from any existing guard[2]; it is then added as a new guard to the graph,
- or it is visible by guards belonging to distinct connected components of the current graph; it is then added as a new connector, and the corresponding connected components are merged,
- otherwise, it is visible only by guards belonging to a same connected component; in such case, it is rejected.

[1] We can note that PRM sacrifices optimisation to efficiency. However, once a path is found, various smoothing and optimisation techniques can be used to improve the solution path

[2] A guard [8] corresponds to a node that is able to access all neighbours by $\mathcal{L}$.

$V \leftarrow \phi$ ; $E \leftarrow \phi$
$Card_E \leftarrow 0$
**While** $Card_E < MAX$ **do**
  $q \leftarrow Random()$
  **If** $q \in CS_{free}$ **Then**
    $V \leftarrow V \cup \{q\}$
    $Card_E \leftarrow Card_E + 1$
    $V_c \leftarrow \{v \in V,$ v neighbour of q $\}$
    **For each** $v \in V_c$(ordered with increasing distance) **do**
      **If** $\neg connect(v, q) \wedge \mathcal{L}(v, q)$ **Then**
        $E \leftarrow E \cup \{(v, q)\}$
    **End** For each
**End** While

**Table 1.** PRM basic algorithm

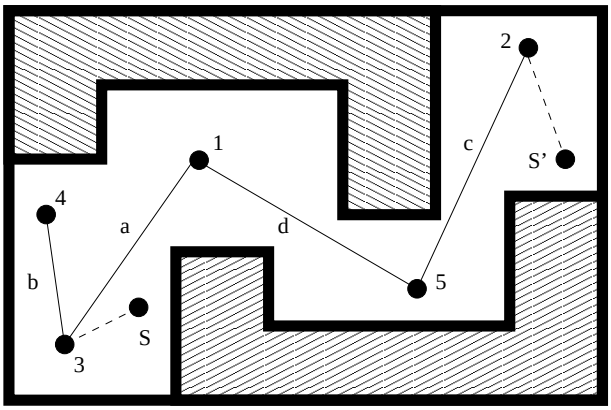


**Fig. 1.** A PRM sequence example

With such algorithm, $c_3$ and $c_4$ of figure 1 would not have been added because of their visibility from $c_1$.

Following the PRM framework, there is clearly a need for a "very efficient" method for testing states connectivity. There is no need here to have a complete planner. Completeness will be ensured by the global planner (PRM approach). The next section proposes an adaptation of GraphPlan that fulfills such a need.

## 3 Adapting GraphPlan to local task planning in large domains

In this section, our aim is to develop a "local task planner" to deal with large problem. This local planner will allow to define the "neighbourhood" notion used in PRM. GraphPlan[1] plans with STRIPS operators and uses a constraint propagation method. It performs in two steps: first, it builds a constraint graph expansion; and then it searches for the plan with a constraint resolution extraction. One limitation comes from the number of possible action instantiations and mutex disappearing for large problems[3].

[3] For this section, we assume that the reader is familiar with GraphPlan algorithms.

For instance, RIFO [10] allows IPP to keep only "relevant" facts and to reduce the state size and possible action instantiations. With such state cuts, IPP is able to plan in quite large domains. Our solution is different: we keep the total state description. However in order to deal with the combinatorial explosion due to action instantiation in large domains, we propose to perform a graph expansion based on partial instantiation and we limit the number of levels. This is reasonable since we are interested in developing a fast local planner (in large domains).

To do so we have defined specific tools for forward expansion and backward search in *partial instantiated action* context.

### 3.1 Partial action instantiation

In STRIPS formalism [4], a state is defined by a set[4] $S$ of positive facts $S = \{f_i\}$. To apply a totally instantiated action $B$ to the state $S$, its precondition set $P = \{p_i\}$ must be included in $S$ ($P \subseteq S$)[5].

Our goal is to reduce the number of developed actions at each level. We decompose each action $B$ in $n$ partially instantiated actions $B_i$ where $n = Card(P)$.

**Definition 1 (Partially instantiated action $B_i$).** *Let $B(\chi)$ be the STRIPS definition of an action $B$ and let $\chi$ its arguments (in first order logic) and $P(\chi)$ its preconditions. Let $\chi_i$ a partial instantiation of $\chi$ such that $p_i(\chi_i)$ is totally instantiated. We call $B(\chi_i)$ (noted $B_i$) a partial instantiation of $B$.*

Note that preconditions and effects of $B_i$ are partially instantiated (except for $p_i$). For instance, table 2 shows a *pick-and-place* action and a partial instantiation by its first precondition.

| pick-and-place( x, y, z) | | pick-and-place( $block_1$, $block_2$, z) | |
|---|---|---|---|
| Precond.: | On(x,y), Clear(x) | Precond.: | On($block_1$,$block_2$), Clear($block_1$) |
| | Clear(z) | | Clear(z) |
| Eff.: | On(x,z), Clear(y) | Eff.: | On($block_1$,z), Clear($block_2$) |
| | ¬ On(x,y), ¬ Clear(z) | | ¬On($block_1$,$block_2$), ¬ Clear(z) |

**Table 2.** (a) An example of STRIPS action in block-world domain. (b) A partially instantiated action (by the first precondition On(x,y) ).

**Definition 2 ($B_i$ Applicability).** *$B_i$ is applicable to state $S$ if and only if $p_i(\chi_i) \in S$ and $\forall j \neq i, \exists \chi_j$ (a total instantiation of $\chi_i$) $p_j(\chi_j) \in S$.*

### 3.2 Mutex propagation

The GraphPlan mutex definition is based on the notion of independence between two actions. Two totally instantiated actions $B^1$ and $B^2$ are independent if $B^1 \circ B^2 \Leftrightarrow B^2 \circ B^1$. The independence relation can be defined by Table 3a properties. We can extend this independence relation to partially instantiated actions. $B_i^1$ and $B_\iota^2$ are independent if $B_i^1 \circ B_\iota^2 \Leftrightarrow B_\iota^2 \circ B_i^1$ with relations defined in Table 3b. Note that the independence relation of partially instantiated actions is weaker than the relation between totally instantiated actions; indeed if $\exists i, \iota$ such that $B_i^1$ is mutex with $B_\iota^2$ then $B^1$ is mutex with $B^2$ but not the converse.

[4] To simplify the notation, we consider states as set of facts instead of conjunctions.

[5] In addition, we have the following properties: i) $D \subseteq P$ and ii) $D \cap A = \phi$ with $D$ (resp. $A$) the set of facts that become false (resp. true) after applying action $B$.

| | | | |
|---|---|---|---|
| $P^1 \subseteq S$ | $P^2 \subseteq S$ | $\forall j, p_j^1(\chi_j) \in S$ | $\forall k, p_k^2(\chi_k) \in S$ |
| $P^1 \cap D^2 = \phi$ | $P^2 \cap D^1 = \phi$ | $\forall j \exists k, p_j^1(\chi_j) \neq D_k^2(\chi_k)$ | $\forall k \exists j, p_k^2(\chi_k) \neq D_j^1(\chi_j)$ |
| $A^1 \cap D^2 = \phi$ | $A^2 \cap D^1 = \phi$ | $\forall j \exists k, A_j^1(\chi_j) \neq D_k^2(\chi_k)$ | $\forall k \exists j, A_k^2(\chi_k) \neq D_j^1(\chi_j)$ |

**Table 3.** (a) Independence between $B^1$ and $B^2$ in $S$. (b) Independence between $B_i^1$ and $B_\iota^2$ ($\chi_\jmath$ (resp. $\chi_k$) is a total instantiation of $\chi_i$ (resp. $\chi_\iota$ ))

**Find_Plan**($\mathcal{G}$,$\mathcal{P}$,$\mathcal{E}_{init}$)
  **If** $\mathcal{G} = \phi$ **Then**
    **Return** *OK*
  Unstack $(X, l_X)$ from $\mathcal{G}$
  **If** $l_X = 0$ **Then**
    **If** $\exists \chi$ an instantiation of X, $\chi \in \mathcal{E}_{init}$ **Then**
      **Return** Find_Plan($\mathcal{G}$,$\mathcal{P}$,$\mathcal{E}_{init}$)
    **Else Return** *Fail*
  **Else**
    Given $\beta$ the set of partially instantiated action that
    supports $X$ and with $l_\beta \leq l_X$
    *Result* $\leftarrow$ *Fail*
    **While** *Result* = *Fail* **do**
      Choose $\{B_i \in \beta\}$such that $X$ is totally instantiated and
      $B_i$ compatible with $B_j, (i \neq j)$ and with $\mathcal{P}$ (non-mutex)
      Add composition of $B_i$ preconditions to $\mathcal{G}$
      Add $\{B_i\}$ to $\mathcal{P}$
      *Result* $\leftarrow$ Find_Plan($\mathcal{G}$,$\mathcal{P}$,$\mathcal{E}_{init}$)
**Return** *Result*

**Table 4.** Plan extraction algorithm for partially instantiated GraphPlan. ($X$ corresponds to a partially or totally instantiated fact, $l_X$ to the level in which we need $X$ and $l_\beta$ the level of $\beta$ appearance.)

Nevertheless using partially instantiated actions allows us to generalise the mutex relation between two facts. In GraphPlan, two facts $P$ and $Q$ are mutex if $P = \neg Q$ or if all actions that support $P$ are mutex with all actions that support $Q$. As we already mentioned, some effects of a partially instantiated action are partially instantiated, and so we can create mutex between two "classes" of facts.

For instance, consider the *pick-and-place* action and a *no-op* action on fact $Clear(block_3)$. The *pick-and-place* action instantiated by the third precondition ($Clear(block_3)$) deletes $Clear(block_3)$ and adds $On(x, block_3)$, whereas *no-op* maintains $Clear(block_3)$. These two actions are mutex, and so we can conclude that $\forall x \in \{block_1, \ldots, block_m\}$, $On(x, block_3)$ is mutex with $Clear(block_3)$.

### 3.3 Solution extraction

Our planner uses the partial action instanciation to expand the mutex graph starting from the initial state. As in GraphPlan, we try to extract a solution as soon as we reach a level that includes the goal. During backward, the planner must find total instantiations for the selected actions. Such a problem has strong similarities with the extensions of GraphPlan to conformant planning (see CGP[13] and [6]). Indeed, we have adapted the algorithm presented in [6] to deal with partially instantiated actions. Table 4 provides a high level description of the plan extraction procedure.

| Block-world domain | | | | | | |
|---|---|---|---|---|---|---|
| | IPP-v4.0 | | | Our planner (PIGP) | | |
| Problem | Level | CPU Time | Memory | Level | CPU Time | Memory |
| 4 blocks | 6 | 0.1 s | 0.4 Mb | 6 | 0.3 s | 2.4 Mb |
| 5 blocks | 8 | 0.2 s | 0.6 Mb | 8 | 2.4 s | 2.5 Mb |
| 6 blocks | 10 | 0.3 s | 0.8 Mb | 10 | 6.5 s | 2.7 Mb |
| 10 blocks | 3 | 5.8 s | 5.7 Mb | 3 | 0.2 s | 2.9 Mb |
| 20 blocks | 3 | 269.3 s | 95 Mb | 3 | 2.9 s | 5.1 Mb |
| 30 blocks | - | >1000 s | >200 Mb | 3 | 5.3 s | 7.9 Mb |
| 100 blocks | - | - | - | 3 | 37.8 s | 15 Mb |

**Table 5.** Results from IPP-v4.0 and our local task planner. The problems (10-100) are defined by: at initial state, all blocks are on table; the goal is to obtain several three block towers.

**Results** Table 5 compares IPP-v4.0 with our GraphPlan adaptation on block-world domain. We note that IPP is significantly faster than our algorithm on small domains. On the other hand our planner can elaborate plans in larger domains when the number of levels necessary to reach the goal is small. Indeed, while the partial instantiation reduces drastically the combinatorial explosion of the graph expansion phase, it is expensive for plan extraction.

This is acceptable in our case since we are interested in developing a fast local planner in large domains.

## 4 Task planning with PRM

In this section, we describe our adaptation of PRM to task planning. It makes use of the local planner defined in the previous section to compute a "topological graph" of the task space. This is done by randomly generating states and trying to connect them to the graph using the local planner.

The process is stopped when we consider that we have a sufficient coverage of the task space. The result is a domain "skeleton", that will be used as a roadmap.

### 4.1 Adaptating PRM to task planning

**Locality and accessibility** Evaluating the distance between two states $d(S, S')$ is NP-Hard. All what we need is an estimation $\delta$ of $d$ with $\delta(S, S') \leq d(S, S')$[6].

In our case, to approximate $\delta$, we use the mutex propagation phase of Graph-Plan. Indeed the number of developed levels to possibly obtain a state $S'$ from $S$ represents the minimal number of independent action set, and so the minimal number of action.

**Definition 3 (Accessibility).** *State $S'$ is accessible from $S$ (noted $\mathcal{A}(S', S)$), if and only if there exists an action sequence $\Gamma$ such that $S' = \Gamma(S)$. The direct accessibility corresponds to the existence of a local plan $\mathcal{L}(S', S)$.*

[6] [2] proposes in HSP to evaluate $\delta$ with the minimal number of actions to obtain $S'$ from $S$ without *delete-liste*. In [3], he estimates cost from the initial state and uses it to define an heuristic from any state.

We note that the accessibility relation is reflexive (i.e. $\mathcal{A}(S,S)$), transitive (i.e. $\mathcal{A}(S,S'') \wedge \mathcal{A}(S'',S') \rightarrow \mathcal{A}(S,S')$) but not necessarily symetric (i.e. $\mathcal{A}(S,S') \not\equiv \mathcal{A}(S',S)$).

**A first algorithm: basic PRM** Table 6 describes the incremental construction of the roadmap. The local planner $\mathcal{L}$ is implemented with a partially instantiated GraphPlan (see previous section) and the distance corresponds to the minimal number of graph levels (here we set the neighboorhood to 3 levels).

For instance, given $G_1$ and $G_2$ two disjoint components of the graph $\mathcal{G}$ and a state $S$. If $S$ is accessible from $G_1$ (i.e. $\exists g \in G_1, \mathcal{A}(S,g)$) and $S$ is not accessible from $G_2$ (i.e. $\neg\exists g \in G_2, \mathcal{A}(S,g)$) then we assume that state $S$ does not provide any new information about the task space topology[7].

Following [12], we define the notion of *Accuracy*; it corresponds to the current number of uninteresting states (since the last interest state). The number $1 - 1/Accuracy$ corresponds to an estimation of the probabilistic coverage of the task space.

**State validity** Our algorithm is based on a random generation of states. While it is quite easy to verify the validity of a given configuration in the path planning domain (no overlapping with obstacles), this is not the case, in general, for task planning. For instance, in the blockword domain, we can not authorize $On(block_1, block_2) \wedge On(block_2, block_1)$. Our planner is not able to check state validity. We assume, that we are able to randomly generate all valid states.

**First results with basic PRM** Figure 2a shows a graph obtained in 7-block-world with a 95% coverage. The program took 727.2 s to build a graph composed of 1867 nodes (3.1 Mb). In this figure, each state is represented by a dot. Two connected dots mean that there exists a local plan between the corresponding states. The position of a given in the diagram depends on the size of highest stack of the state (radius of the circles from 1 to 7) and the number of the first block of the stack (angle of the supporting segment). We note that the figure is strongly symmetric, especially because of the 7 possible first blocks. Indeed, in block-world domain, for a tower of $n$ blocks, there are $n!$ possible configurations.

Figure 2b presents a graph built for the 3-mail problem with a 95% coverage: a robot must move letters from a table to another in a complex environment. In this example, the environment contains 400 cells which are connected with 160000 facts (e.g. $connect(c_{1,1}, c_{1,2})$). Tables are represented by grey cells and walls by black cells. Our algorithm used 18 Mb and took 386.4 s to build a graph composed of 131 nodes. Note that the position of the nodes depends only on robot position and not on the position of the letters (letters can be left on tables or on the robot). This is the reason why there are a number of neighbour nodes on the figure which are not connected.

From these two (non trivial) examples, we can make two observations. First, our algorithm successfully "captured" a topological structure of the task space derived

[7] We assume that $S$ will again be randomly generated, in the future, to test again possible connections between connected components of $\mathcal{G}$. That is the reason why, we can say that the probabilistic completeness of the method is ensured.

$\mathcal{G} \leftarrow \{\phi\}$
$Accuracy \leftarrow 0$
**While** $Accuracy < MAX$ **do**
  $S \leftarrow RandomValidState()$
  $found_G \leftarrow \phi$ ; $found_g \leftarrow EmptyState()$
  $found_{nb} \leftarrow 0$
  **For each** $G \in \mathcal{G}$ (ordered with increasing distance) **do**
    **If** $\exists g \in G, \mathcal{A}(S, g)$ **Then**
      **If** $found_{nb} = 0$ **Then**
        $found_G \leftarrow G$ ; $found_g \leftarrow g$
        $found_{nb} \leftarrow 1$
      **Else**
        **If** $found_{nb} = 1$ **Then**
          Connect $S$ to $found_G$ by $found_g$
        $\mathcal{G} \leftarrow \mathcal{G} - G$
        Connect $S$ to $G$ by $g$
        $found_{nb} \leftarrow found_{nb} + 1$
  **End** For each
  **If** $found_{nb} = 0$ **Then**
    $\mathcal{G} \leftarrow \mathcal{G} \cup \{S, \phi\}$
  **If** $found_{nb} = 1$ **Then**
    $Accuracy \leftarrow Accuracy + 1$
  **Else**
    $Accuracy \leftarrow 0$
**End** While

**Table 6.** Accessibility based algorithm

from accessibility by local plans. Second, in both domains there are symmetries and possible permutations which unusefully increase the graph.

### 4.2 A PRM that deals with permutations

Due to all possible permutations between different states, the previous method to build a task topological graph is not able to capture the domain topology with a polynomial number of states (see for instance the symmetry that appears in figure 2a). To solve this problem, we propose an extension that deals explicitly with permutations.

For example, when there is a permutation between two states $S_1$ ($On(block_1, block_2) \wedge OnTable(block_2)$) and $S_2$ ($On(block_2, block_1) \wedge OnTable(block_1)$), we will try to learn the task space topology for only one permutation. In this case, the environment "skeleton" is $On(X, Y) \wedge OnTable(Y)$ and there are two possible substitutions $\sigma(S_1, S_2)$ $\{X/block_1; Y/block_2\}$ and $\sigma(S_2, S_1)$ $\{X/block_2; Y/block_1\}$.

We define $\mathcal{A}^+$ the accessibility relation $\mathcal{A}$ augmented by substitution which is transitive: Given $S_1$, $S_2$ and $S_1'$ three states such that $\mathcal{A}(S_1, S_2)$, $\mathcal{A}(S_2, S_1)$ and $\sigma(S_1, S_1')$. In that case, there is a plan $\Gamma$ to connect $S_2$ to $S_1$. Given $\Gamma'$, the plan $\Gamma$ modified by the substitution $\sigma(S_1, S_1')$, and $S_2'$ the result of the substitution

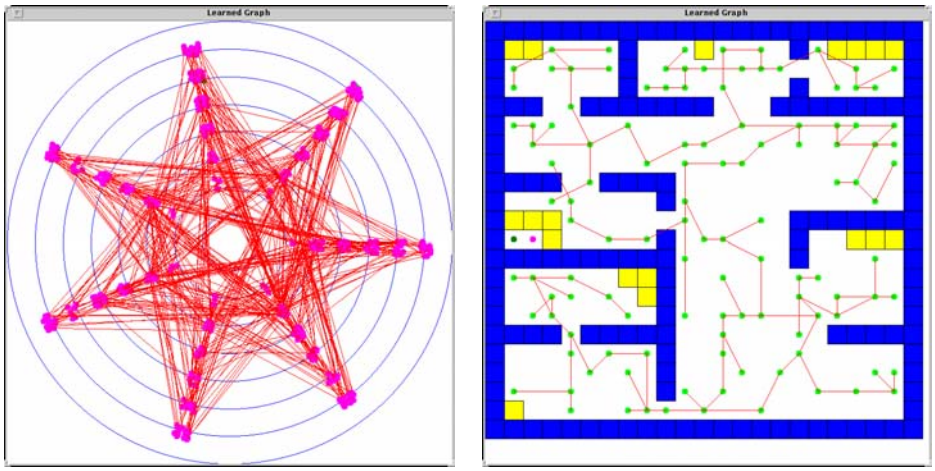


**Fig. 2.** (a) Learned (95%) graph in 7-block-world domain. (b) Learned (95%) graph in 3-Mail domain.

$\sigma(S_1, S_1')(S_2)$ ; we have $S_1' = \Gamma'(S_2')$ and we conclude that the accessibility relation $\mathcal{A}(S_1', S_2')$ is valid. So if we have $\mathcal{A}(S_2', S_2)$ then we can deduce $\mathcal{A}(S_1', S_1)$ (*via* the path $S_1 \rightarrow S_2 \rightarrow S_2' \rightarrow S_1'$).

Consider for instance the 3-block-world domain. Given $S_1 = \{\, On(block_1, block_2), On(block_2, block_3), On\_table(block_3) \,\}$, $S_1' = \{\, On(block_3, block_1), On(block_1, block_2), On\_table(block_2) \,\}$ and $S_2 = \{\, On(block_2, block_3), On\_table(block_3), On\_table(block_1) \,\}$. We note that $\neg\mathcal{L}(S_1, S_1')$, $\mathcal{L}(S_1, S_2)$ and $\mathcal{L}(S_2, S_1)$. In addition, we can reach $S_2'$ from $S_2$ in two steps, so we can conclude that $\mathcal{A}^+(S_1', S_1, \sigma(S_1, S_1'))$[8]

$\mathcal{A}^+(S_1', S_1, \sigma_{1\rightarrow 1'})$
  **If** $\mathcal{L}(S_1, S_1')$ **Then**
    **return** *OK*
  **For each** $S_2$ such that $\mathcal{L}(S_1, S_2) \wedge \mathcal{L}(S_2, S_1)$ **do**
    $S_2' \leftarrow \sigma_{1\rightarrow 1'}(S_2)$
    **If** $\mathcal{A}^+(S_2', S_2, \sigma_{1\rightarrow 1'})$ **Then**
      **return** *OK*
  **End** For each
  **return** *Fail*

**Table 7.** Accessibility based on permutation

This property allows an extension of the basic PRM algorithm that takes into account substitution. The new algorithm is similar to the algorithm presented in table 6 but, instead of using $\mathcal{A}$ to test the accessibility of a new random state $S_1$, we test if it corresponds to a permutation between two components $G_1$ and $G_2$ of the graph $\mathcal{G}$ ($S_2 \in G_2$ such that $\sigma(S_1, S_2)$). If it is the case, we store $S_1$, $S_2$ and $\sigma(S_1, S_2)$ and use such permutation to try to connect $G_1$ and $G_2$ in the

[8] We note that $\mathcal{A}(S_1', S_1)$ is false because our local planner $\mathcal{L}$ is limited to three steps.

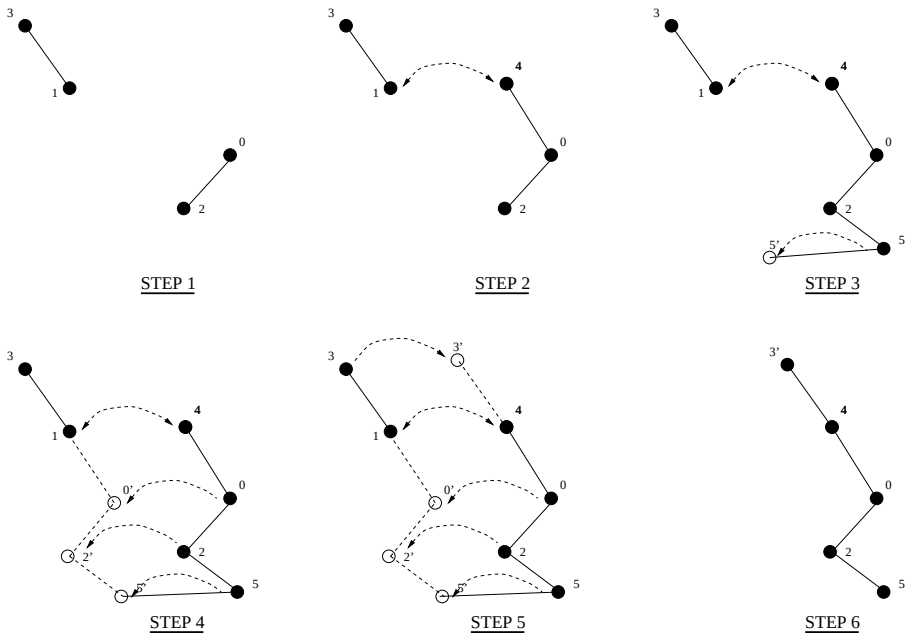


**Fig. 3.** A PRM sequence exemple

subsquent steps. Besides, if the connection is established, we check the graph in order to eliminate redundancies (see step 6 of figure 3).

The following example illustrates the overall process[9] (see figure 3). At step 1, there are two components in our graph ($S_0 - S_2$ and $S_1 - S_3$). Solid line represent accessibility. At step 2, we randomly generate $S_4$. $S_4$ is accessible from $S_0$ because of $\mathcal{L}$. In addition, we note that there is a substitution $\sigma(S_4, S_1)$. Dashed curve denote a substitution. So the question is: can we connect components $S_4 - S_0 - S_2$ and $S_1 - S_3$? At this step, it is impossible because of: $\neg\mathcal{A}(S_1, S_4)$, $\neg\mathcal{A}(S_3, S_4)$ and not connected at step 1, then we deduce $\neg\mathcal{A}^+(S_1, S_4)$. Now we store $\sigma(S_4, S_1)$ to check if further states can connect the two components via $\mathcal{A}^+$ (this is the case with $S_5$ and $S_{5'}$). At step 3, we randomly generate $S_5$. $S_5$ is accessible from $S_{5'}$ ($S_{5'}$ is created with $\sigma(S_4, S_1)$. In this case (see step 4)), $S_4$ is accessible from $S_1$ (indeed we have $S_1 - S_{0'} - S_{2'} - S_{5'} - S_5 - S_2 - S_0 - S_4$). Dashed line means that states are already connected in the other substitution. Now, we can reduce the graph and create only one component without substitution. At step 5, we use $\sigma(S_4, S_1)$ to create $S_{3'}$ from $S_3$. At step 6, we delete $S_1$ (resp. $S_3$) which is now accessible from $S_4$ (resp. $S_{3'}$) *via* $S_5$.

**Block-world results** Figure 4a shows a learned graph for the 7-block-world, with a 99% coverage, domain with permutation method. Our algorithm uses 4.0 Mb during 7.9 s to find 14 nodes Each dot is labelled by a state number described

[9] In order to simplify the explanation we assume that a state accessible from only one component is added to $\mathcal{G}$. For instance in step 1, nodes 2 and 3 are added. We also assume that $\mathcal{A}$ is symmetric.

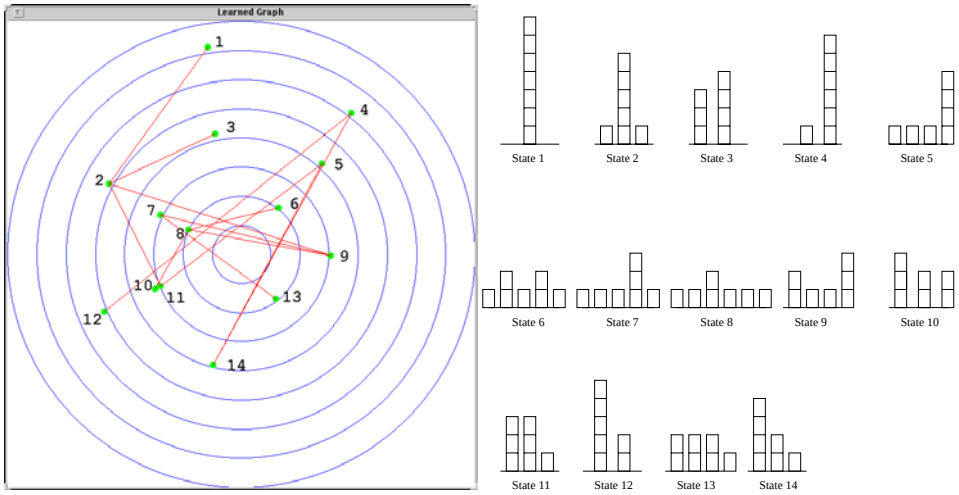


**Fig. 4.** Learned (99%) graph with substitution method in 7-block-world domain

in figure 4b. We note that for 7 blocks our graph contains only 14 states. These states correspond, in fact, to "classes" of states; indeed because of the permutation reasoning a state of this graph is not really instantiated but represents a whole class of states obtained by substitution.

### 4.3 Solution search

The solution extraction method is similar to the insertion of one state during the learning phase. Indeed, we must connect the initial state $S_{init}$ to $S_i \in \mathcal{G}$, connect the goal $S_{goal}$ to $S_g \in \mathcal{G}$ and then find a path[10] between $S_i$ and $S_g$ when $\mathcal{A}^+(S_i, S_g)$[11].

Table 8 shows some results on block-world domain (with permutation). Results from 7-block-world express permutation reasoning capabilities: 7.9 s / 14 nodes with 99% *vs.* 727.2 s / 1827 nodes with 95%. In addition, we remark that if we remove the average time spent to connect initial and goal state to the graph from the average time to extract a solution, the spent time to find a path is about 0.1 s.

## 5 Conclusion and future work

We have proposed an extension of probabilistic Roadmap Methods (PRM) to task planning. Such an extension can not be reasonably attempted without an efficient local planner which may answer quickly to "connections" requests.

This is why we have developed an extension of GraphPlan[1] specially designed to deal with "local planning" in large domains. It is essentially based on the construction of mutex between partially instantiated facts.

[10] We note that the planner is sound. Indeed the solution is extracted from the graph and the connection between the initial and final state; all connections are built by the GraphPlan extension (sound too); so if a path exists, it is consistent.

[11] If we can not connect $S_i$ or $S_g$ we can use a classical planner (e.g. IPP-v4.0).

| Problem | Graph learning $nb_{node}$ | CPU Time | Memory | Solution search CPU Time (average) |
|---|---|---|---|---|
| 7 blocks | 14 | 7.9 s | 4.0 Mb | 0.07 s |
| 10 blocks | 31 | 44 s | 4.8 Mb | 0.31 s |
| 15 blocks | 86 | 492.5 s | 5.6 Mb | 1.9 s |
| 20 blocks | 253 | 4186.8 s | 6.6 Mb | 5.4 s |

**Table 8.** Learn and solution extraction phase on block-world domain

Another key feature is the development of techniques that allow to reduce as much as possible the size of the learned graph without "losing" the probabilistic completeness.

This research is still preliminary. However, the obtained results are very promising. Our future work will concern further investigations on the following aspects: i) improvement of the local planner efficiency (for example, can we introduce some heuristics [3]? ii) improvement of the topological structure identification by adding more general symmetry analysis[5], or extending re-using methods ([11], [7], ...).

## References


1. A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, pages 281–300, 1997.
2. B. Bonet and H. Geffner. Hsp: Planning as heuristic search. *http://www.ldc.usb.ve/hector*, 1998.
3. B. Bonet and H. Geffner. Planning as heuristic search: New results. *5th European Conference on Planning (ECP'99)*, 1999.
4. R.E. Fikes and N.L. Nilsson. Strips: A new approach to the application of therom proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
5. M. Fox and D. Long. The detection and exploration of symmetry in planning problems. *Proc. 16th Inter. Joint Conf. on Artificial Intelligence (IJCAI'99)*, 1999.
6. 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.
7. S. Kambhampati and J.A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55:193–258, 1992.
8. L. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57:50–60, 1998.
9. L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transaction on Robotics and Automation*, 12:566–580, 1996.
10. B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *4th European Conference on Planning (ECP'97)*, 1997.
11. B. Nebel and J. Koehler. Plan reuse versus plan generation: A theorical and empirical analysis. *Artificial Intelligence*, 76:427–454, 1995.
12. 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.
13. D. Smith and D. Weld. Conformant graphplan. In *Proc. 15th Nat. Conf. AI.(AAAI'98)*, 1998.