One action is enough to plan

Emmanuel Guere, Rachid Alami

One action is enough to plan

Emmanuel Guéré and **Rachid Alami**
LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex 4 - France
e-mail : {Emmanuel.Guere, Rachid.Alami}@laas.fr

Abstract

We describe a new practical domain independent task planner, called ShaPer, specially designed to deal efficiently with large problems.

ShaPer performs in two steps. In the first step, executed *off-line* for a given *domain subclass*[1], ShaPer explores and builds a compact representation of the state space called the *shape* graph. The main contribution of ShaPer is its ability to "resist" to combinatorial explosion thanks to the manipulation of sets of similar state descriptions called *shapes*. The *shape* graph is then used by ShaPer to answer very efficiently to planning requests.

A first version of the planner has been implemented. It has been tested on several well known benchmark domains. The results are very promising when compared with the most efficient planners from AIPS-2000 competition.

1 Introduction

In the stream of research that aims to speed up practical task planners, we propose a new approach to efficiently deal with large problems. Even though task planners have made very substantial progress over the last years, they are still limited in their use. Indeed, they are sometimes overwhelmed by very simple or even trivial problems. Our motivation stems also from the fact that there are domains which heavily influence the "structure" of the task state space; learning such a "structure" will certainly help in building efficiently a solution for a problem of the *domain* $subclass$[1]. Our aim is to develop a domain independent planner that will exhibit and learn the "structure" of a given *domain subclass*. The main difficulty (and the key contribution) in this framework is to face the combinatorial problem of the task space exploration.

In order to solve larger problems, all planners in the literature try to prune the state space. To do this, there are several ways: a first solution consists in having a "good metrics" of the domain in order to guide the search (heuristics). The heuristics can be automatically computed (domain independent planning - e.g. [Bonet and Geffner, 1999; Hoffmann, 2000]) or given by the user (domain dependent planning - e.g. [Doherty and Kvarnstrom, 1999]).

A second way consists in restricting the search to a superset of accessible states from the initial state; such restriction allows to reduce the search space during the backward planning process (e.g. the graph expansion of GraphPlan [Blum and Furst, 1997]).

A third way involves the analysis of the domain "structure" like invariant (e.g. [Gerevini and Schubert, 1998]), type detection, problem decomposition or problem symmetries (e.g. [Fox and Long, 1998; 1999]).

We would like is to develop a new, complete and domain independent method to efficiently solve large problems. To do so, we both restrict the search space through a set of accessible states and deal with a large class of domain symmetries (represented by states which have the same *shape*[2]).

For instance, the ferry problem overwhelms classical planners because of the very high number of possible applicable actions (which may create many redundant states) even though moving one car or another has the same result for the goal. However, it appears clearly that the state space is highly redundant: there is only $2n + 1$ distinguishable *shapes* for n cars. Dealing with such *shapes* will certainly increase the planner capabilities by reducing the search space in a drastic manner.

ShaPer[3] is a new planner which is able to detect all different *shapes* of the state space for a given *domain subclass*. This method is composed of two steps: first the planner builds a *shape* graph of the *domain subclass*. This step is performed *off-line* and only once for a given *domain subclass*, *e.g.* ShaPer builds the *shape* graph for the *subclass* of ferry domain with 50 cars. Then, ShaPer is able to solve *on-line* any planning problem in this class by connecting two *shape* graphs. As we will see, the connection makes use of only **one action**; this ensures the efficiency of the solution extraction step.

The next section explains how to build such *shape* graph. Section 3 presents an expansion algorithm that guarantee

[1] In this paper, we call a *domain subclass* the set of planning problems defined by a set of operators and a set of objects. For example, the 10-blocks-world is a *subclass* of the blocks-world domain. The 20-blocks is another *subclass* of the same domain.

[2] The *shape* of a state, defined more precisely in the next section, can be viewed as a partially instantiated state pattern.

[3] *Sha*pe based *P*lann*er*

ShaPer completeness. We then describe the solution extraction process (section 4). A first version of the planner has been implemented and has produced very promising results. All sections are illustrated by examples obtained by running such an implementation. The last section presents and discusses a number of tests which are compared with some of the best current planners (from AIPS-2000 competition).

2 The Shape Graph: $\mathcal{G}$

A planning problem is usually expressed as a triple $(\mathcal{O}, \mathcal{I}, \mathcal{J})$ where $\mathcal{O}$ corresponds to the set of instantiated actions, $\mathcal{I}$ the initial state and $\mathcal{J}$ the goal. In the STRIPS [Fikes and Nilsson, 1971] formalism, an action $o \in \mathcal{O}$ is described by its *preconditions* P_o ($P_o \subseteq S$ means that o is applicable to the state S), its *Addlist* A_o and its *Dellist* D_o. Applying the operator o from state S results in the new state $o(S) \equiv (S - D_o) \cup A_o$. A state is an instantiated predicate set according to the closed-world assumption.

The purpose of this section is to present the *shape* graph construction process (only extract "relevant" states). This state space exploration is performed *off-line* and once only for each *domain subclass* from a valid state S_{begin}.

Indeed, if we restrict ourselves to the STRIPS framework, there is *a priori* no mean to check if a state is valid or not, except by applying a valid sequence of action to a valid one (S_{begin} - user-defined).

This is the reason why the *shape* graph is an accessibility graph $\mathcal{G}$. Now, the main difficulty comes from the combinatorial explosion of states and applicable actions; this is why we build a graph that only contains "relevant" states.

2.1 Relevant states

The relevance of a state is defined according to the state description of the current graph $\mathcal{G}$. A state S is relevant (i.e. it "augments" $\mathcal{G}$ with new information) iff there exists no $g \in \mathcal{G}$ such that g is a substitution σ of S: $\sigma(g) = S$ (i.e. the state g can be instantiated by a variable permutation σ). Consider for instance the two blocks-world states g and S:

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 where A is substituted to A, B to C and C to B is equal to S ($\sigma = \{A/A, C/B, B/C\}$). In this case, g and S are said to have the same *shape*[4].

When S has the same *shape* as g, we can conclude that all *shapes* accessible from g are also accessible from S: if P is a sequence of action applicable to g, then $\sigma(P)$ is applicable to S and $(\sigma(P))(S) = \sigma(P(g))$. Developing the state S is then not informative: S does not lead to a new *shape*.

[4]In order to prevent possible misunderstandings due to the previous figure, let us emphasize on the fact that a *shape* does not characterize a tower, but more generally the structure of a planning problem (i.e. there exists a substitution between S and g).

Build_Graph($\mathcal{O}$,S_{begin})
  $To\_Dev \leftarrow \{S_{begin}\}$
  $\mathcal{G} \leftarrow \{S_{begin}\}$
  While $To\_Dev \neq \phi$ **do**
    $s \leftarrow pop(To\_Dev)$
    For each $o \in \mathcal{O}$ such that $P_o \subseteq s$ **do**
      If $\exists g \in \mathcal{G}$ and $\exists \sigma$ such that $o(s) = \sigma(g)$ **Then**
        If $o(s) = g$ **Then**
          Add the edge (s, g) to $\mathcal{G}$
        Else
          Mark s in $\mathcal{G}$ with σ and g
      Else
        Add the vertex $o(s)$ and the edge $(s, o(s))$ to $\mathcal{G}$
        $To\_Dev \leftarrow To\_Dev \cup \{o(s)\}$

Table 1: Build the *shape* graph $\mathcal{G}$.

2.2 Building the shape graph $\mathcal{G}$

In order to reduce the graph size, we only develop relevant states. The algorithm, presented in table 1, is similar to a breadth-first search algorithm. Relevant states are sequentially developed by applying all possible actions o. The algorithm detects links to other substitutions as well as cycles in a given substitution (when $o(s) = g$); in this case, a new edge is added.

In the case of a non-identical substitution σ, we keep in s the name of the corresponding *shape* in $\mathcal{G}$ and σ. Owing to the relation $o(s) = \sigma(g)$, the accessibility from s to $o(s)$ can be, if necessary, quickly retrieved. Indeed, several substitutions may produce the same result because of the commutativity of the logical and, *i.e.* $\sigma(g) = \sigma'(g)$ does not imply $\sigma = \sigma'$.

2.3 An example

The graph construction algorithm is illustrated by figure 1 with an example from the gripper domain. The goal is to move 3 balls from table T_1 to table T_2. To do this, the robot is able to pick and place a ball and to move from a table to the other. The robot has two arms. The graph expansion begins with the valid state S_1 (all three balls are on T_1 and the robot is near T_1). Four actions are applicable to S_1: $Pick(X, T_1)$, $Pick(Y, T_1)$, $Pick(Z, T_1)$ and $move(T_1, T_2)$; as shown on figure 1 $Pick(X, T_1)(S_1) = S_2$ and $move(T_1, T_2)(S_1) = S_3$. We can note that $Pick(Y, T_1)(S_1)$ is not added to the graph, since there exists a substitution $\sigma = \{Y/X, X/Y, Z/Z, T_1/T_1, T_2/T_2, g_l/g_l, g_r/g_r\}$[5] such that $\sigma(S_2) = Pick(Y, T_1)(S_1)$. Similarly, there exists a substitution between $Pick(Z, T_1)(S_1)$ and S_2; we note it on the figure by a dashed line to S_2. Then S_2 is developed in S_4 and S_5 and so on ...

The graph $\mathcal{G}$ finally contains nine vertices which model all possible *shapes* accessible from the valid state S_1.

[5]Note that in this example, even through the substitution only permutes balls X, Y and Z, the substitution σ must contain g_l (gripper_left), g_r (gripper_right) and the two tables T_1 and T_2 because they are variables too. See for instance the state S_9 which has the same *shape* as the state S_8 when the robot takes a ball.

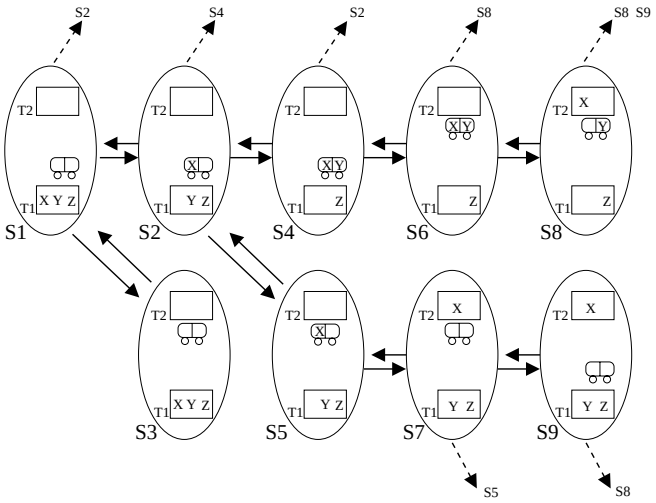

Figure 1: Build the *shape* graph $\mathcal{G}$.

2.4 $\mathcal{G}$ and the state space

This method makes it possible to drastically reduce the size of the state space when the domain contains many (functionally similar) "objects" (variables with possible substitutions). Indeed, in the blocks-world and in the gripper domain, the state space grows exponentially comparing to the *shape* graph size as shown table 2.

As mentioned previously, ShaPer needs a valid state, S_{begin}, to build the graph $\mathcal{G}$. Although $\mathcal{G}$ contains all *shapes* accessible from S_{begin}, in the case of a disjoint state space, ShaPer is unable to construct graphs from the other connected components of state space (their states are not accessible). Therefore, the user must give one state per connexity; otherwise, the *shape* of S_{init} (the initial state of a planning problem) might not be present in $\mathcal{G}$, which will constraint the planner to generate a complementary *shape* graph from S_{init} during the *on-line* process.

2.5 A first step through solution extraction

To perform efficient *on-line* solution extraction, ShaPer takes advantage of the *shape* graph $\mathcal{G}$ built *off-line*. Three steps are necessary to find a plan: first, generate the graph $\mathcal{G}_{init}$ (resp.

problem	state space size	$\mathcal{G}$ size
blocks-world		
3 blocks	13	3
4 blocks	73	5
5 blocks	501	7
6 blocks	4051	11
7 blocks	41413	15
gripper		
3 balls	88	9
4 balls	256	12
5 balls	704	15
6 balls	1856	18

Table 2: Growth of the state space comparing to the $\mathcal{G}$ size.

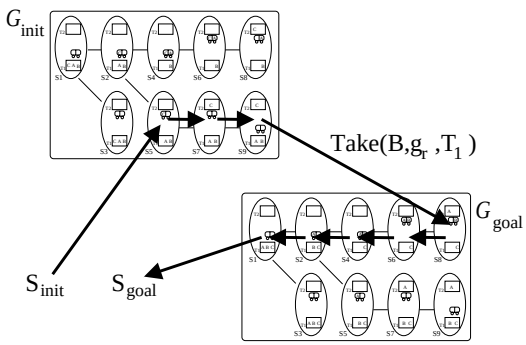

Figure 2: Extract a solution by connecting $\mathcal{G}_{init}$ and $\mathcal{G}_{goal}$.

$\mathcal{G}_{goal}$) associated to the initial state S_{init} (resp. S_{goal}). Then ShaPer searches for **one** action $o \in \mathcal{O}$ that connects $\mathcal{G}_{init}$ with $\mathcal{G}_{goal}$ *via* s_i and s_g ($s_i \in \mathcal{G}_{init}$,$s_g \in \mathcal{G}_{goal}$ and $s_g = o(s_i)$). Finally it is enough to find a path from S_{init} to s_i in $\mathcal{G}_{init}$ and from s_g to S_{goal} in $\mathcal{G}_{goal}$.

To better understand this intuitive algorithm, let us explain it through the 3-balls-gripper example. There are three balls, two tables and two grippers; initially balls A and B are on table T_1 and the ball C is in the left gripper near table T_2. This state has the same *shape* as S_5 of the figure 1 where $\{C/X, A/Y, B/Z\}$. The goal is to obtain the three balls on T_2. To do so, we instantiate the *shape* graph $\mathcal{G}$ with the initial state substitution (resp. goal) and obtain the graph $\mathcal{G}_{init}$ (resp. $\mathcal{G}_{goal}$). Then the algorithm looks for one action which connects a state of $\mathcal{G}_{init}$ to a state of $\mathcal{G}_{goal}$, as illustrated by figure 2 with the action $Take(B, g_r, T_1)$.

3 $\mathcal{G}$ and completeness

The method, for solution extraction (proposed in the previous section) is generally very efficient, however in specific case, it is not always possible to connect the two graphs with a single action. To obtain a solution, the planner may need to connect several substitutions of the graph $\mathcal{G}$; but looking for such transitions may be as costly as planning "from scratch".

Fortunately, we can compute *off-line* a *meta-graph* $\mathcal{H}$ which contains all necessary substitutions of $\mathcal{G}$ to solve any instance of the problem (from the learned domain) in only one action (between $\mathcal{H}_{init}$ and $\mathcal{G}_{goal}$).

3.1 $\mathcal{H}$: a meta-graph of $\mathcal{G}$

In order to ensure the completeness for our method, we propose an expansion algorithm of the graph $\mathcal{G}$ to $\mathcal{H}$. The main idea is the following: if all substitutions accessible from S_{begin} are accessible from $\mathcal{H}$ too, then there exists an action $o \in \mathcal{O}$ which connects $\mathcal{H}_{init}$ to $\mathcal{G}_{goal}$.

To perform such expansion, we begin with $\mathcal{H} = \mathcal{G}$. $\mathcal{H}$ must contain all graph $\mathcal{G}$ that allow to connect graph $\mathcal{G}'$ such that $\mathcal{G}'$ is not directly connected to $\mathcal{H}$. For each $o(h)$ (with $h \in \mathcal{H}$ and $o(h) \notin \mathcal{H}$), generate the graph $\mathcal{G}_{o(h)}$. For each $o'(g)$ (with $g \in \mathcal{G}_{o(h)}$ and $o'(g) \notin \mathcal{G}_{o(h)}$), check if $\mathcal{G}_{o'(g)}$ is directly accessible from $\mathcal{H}$; if it is not the case, then add $\mathcal{G}_{o(h)}$ to $\mathcal{H}$. This algorithm stops when $\mathcal{H}$ becomes invariant. In other words, $\mathcal{H}$ corresponds to the "transitive closure" of $\mathcal{G}$ in terms of substitutions.

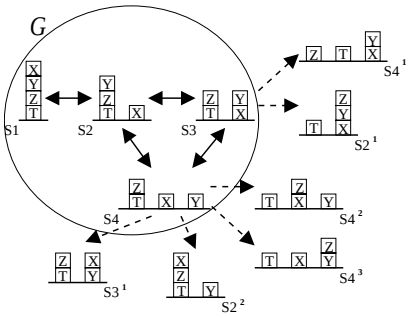

Figure 3: The graph $\mathcal{G}$ for 4-blocks-world domain restricted to 3 stacks.

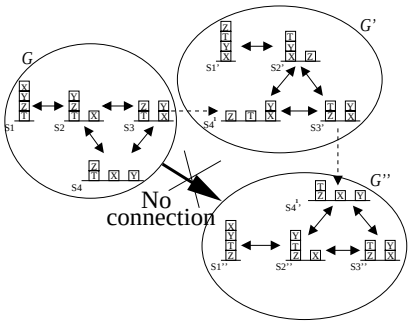

Figure 4: Expanding the graph $\mathcal{G}$ to ensure completeness

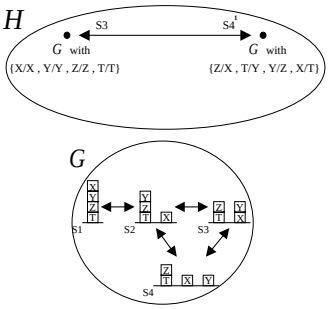

Figure 5: $\mathcal{H}$ models the state space accessible with **one** action from $\mathcal{G}$

3.2 An expansion example

To better understand this expansion, we illustrate it on the 4-blocks-world domain restricted to three stacks. Figure 3 presents the *shape* graph $\mathcal{G}$ and its substitutions (e.g. S'_4 is a substitution of the state S_4). Intuitively, to reverse blocks Z and T in this domain, it is necessary to use two substitutions of S_4; so find a solution with only one action may be impossible only by using $\mathcal{G}$.

To expand $\mathcal{G}$ presented in figure 3, ShaPer must examine states S_4^1, S_2^1, S_4^2, S_4^3, S_2^2 and S_3^1 (states which leave $\mathcal{G}$). Developing $\mathcal{G}'$, the *shape* graph from S_4^1, allows to generate $\mathcal{G}''$ *via* $S_4^{1\prime}$ (see figure 4). Note that there does not exist any action to connect a state of $\mathcal{G}$ to a state of $\mathcal{G}''$; this means that $\mathcal{G}'$ allows to access to new substitutions. Consequently, it is added to $\mathcal{H}$. The graph presented in figure 5 corresponds to the invariant of $\mathcal{H}$ after having examined all the other states.

Naturally, all the graphs included in $\mathcal{H}$ differ only by a substitution. Therefore, $\mathcal{H}$ can be expressed by only using substitutions and $\mathcal{G}$, *e.g.* in figure 5, $\mathcal{H}$ is described by two substitutions on $\mathcal{G}$.

4 Solution extraction

$\mathcal{G}$ and $\mathcal{H}$ being computed *off-line*, ShaPer performs *on-line* plan extraction for any possible problem from the learned domain by searching for **one** action.

Extract_Solution($\mathcal{O}$,$\mathcal{G}$,$\mathcal{H}$,S_{init},S_{goal})
Find σ and $g \in \mathcal{G}$ such that $S_{init} = \sigma(g)$
$\mathcal{H}_{init} = \sigma(\mathcal{H})$ /* Instantiate $\mathcal{H}$ with σ */
$\mathcal{G}_{goal} = \theta(\mathcal{G})$ with $S_{goal} = \theta(g')$
$cost \leftarrow 0$
While not examine all states of $\mathcal{H}$ **do**
For each substitutions τ of $\mathcal{H}$ **do**
For each triple (s, g'', μ) of $\tau(\mathcal{G})$ **do**
/* $\exists o \in \mathcal{O}.o(s) = \mu(g'')$ μ a substitution */
/* and $(s, g'') \in \mathcal{G}^2$ */
If $C(S_{init} \rightarrow s) + 1 + C(o(s) \rightarrow S_{goal}) = cost$
Then If $o(s) = \theta(g'')$ **Then**
return the plan: Plan($S_{init} \rightarrow s$), o, Plan($o(s) \rightarrow S_{goal}$)
$cost \leftarrow cost + 1$
return: No solution

Table 3: Algorithm for solution extraction

4.1 An efficient algorithm

Table 3 presents the three-steps algorithm used to extract a plan from an initial state S_{init} to S_{goal}. As the graph $\mathcal{G}$ is computed *off-line*, it is also possible to compute the best path for any couple of vertices in $\mathcal{G}$. Let $Plan(g_1, g_2)$ be the optimal plan from g_1 to g_2 in $\mathcal{G}$ and $C(g_1, g_2)$ the number of actions of this plan. Considering the cost C, it is possible to find the best solution using only two *shape* graphs and one action. Indeed, the algorithm examines iteratively all possible connections, ordered by an increasing cost (the cost is defined by $C(S_{init}, s) + 1 + C(o(s), S_{goal})$). Such procedure makes it possible to obtain non-optimal, because of the graph learning, but good solutions. This is illustrated by the number of plan steps in the results presented in table 4.

Figure 6 shows an example of a solution extraction in 4-blocks-world domain restricted to three stacks. First, ShaPer instantiates the graph $\mathcal{H}$ with the initial state ($\sigma_{init} = \{D/X, B/Y, A/Z, C/T\}$) obtaining $\mathcal{H}_{init}$, and $\mathcal{G}$ with the goal ($\sigma_{goal} = \{D/X, C/Y, B/Z, A/T\}$) obtaining $\mathcal{G}_{goal}$. Then, in order to find a connection between $\mathcal{H}_{init}$ and $\mathcal{G}_{goal}$, the algorithm examines (following an increasing cost) the states $S_4^1 \ldots S_3^1$ and $S_4^{1\prime} \ldots S_3^{1\prime}$ (see figure 3) and finds $S_4{}' \rightarrow S_4^{3\prime}$ as first connection. Then, to build a valid

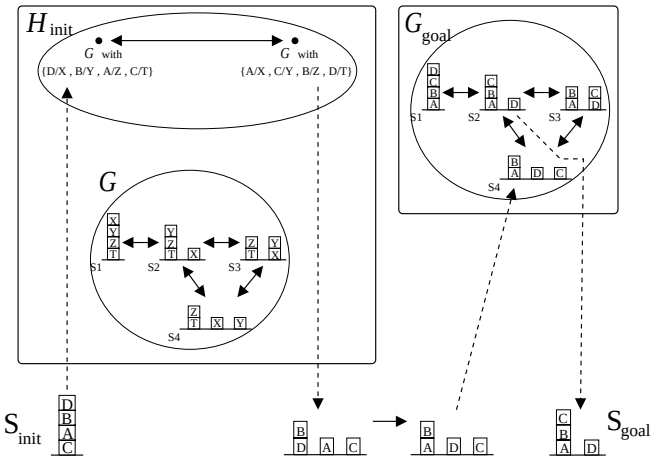

Figure 6: A solution extraction example in 4-blocks-world domain restricted to three stacks

plan, it is enough to find a plan in $\mathcal{H}_{init}$ and a plan in $\mathcal{G}_{goal}$. Thus, the plan extracted by ShaPer is the following: [*MoveToTable(D,B), Move(B,A,D), MoveToTable(A,C)*], *Move(B,D,A)*, [*MoveFromTable(C,B)*].

4.2 Bounds for Computation Time and plan length

Having the *shape* graph computed *off-line* leads to an interesting property: the solution extraction algorithm is time bounded and the plans have a known maximal length.

Indeed, the algorithm presented in table 3 examines iteratively a set of triples (s, g'', μ) of $\tau(\mathcal{G})$. The test consists in checking equality between two states. As $\mathcal{H}$ is computed *off-line*, we know in advance how many triples there are, *e.g.* for 15 blocks there are only 7186 triples. Consequently, we can compute an upper limit for the *on-line* solution extraction computation time.

In the same way, we can compute an upper limit n for the longest path P in $\mathcal{G}$. The length of the longest plan (maximum number of plan steps) will then be bounded by $2n + 1$.

The results, presented in table 4, include a *max* column for number-of-steps and computation time bounds.

4.3 Completeness

This subsection present the main ideas to prove the completeness of ShaPer in informal way.

For a substitution σ, $\sigma(A \cup B) = \sigma(A) \cup \sigma(B)$ and $\sigma(A \cap B) = \sigma(A) \cap \sigma(B)$. So we can demonstrate the following theorem: $\sigma(o(s)) = (\sigma(o))(\sigma(s))$ with o being an action applicable to the state s[6].

The algorithm to build $\mathcal{G}$ ensures that for any state s in $\mathcal{G}$ and any action o such that $o(s) \notin \mathcal{G}$, there exists a substitution σ of a state g of $\mathcal{G}$ with $o(s) = \sigma(g)$. Then it is possible to demonstrate that $\mathcal{G}$ contains all accessible *shapes*. Given $s \in \mathcal{G}$ and $e = o(s) \notin \mathcal{G}$. Suppose that e allows to access to the state $s' = o'(e)$ which has a new *shape*. From $\mathcal{G}$ definition, $\exists \sigma \wedge g \in \mathcal{G}.\sigma(g) = e$. The action $\sigma^{-1}(o')$ is applicable to g; if $\sigma^{-1}(o')(g) \notin \mathcal{G}$ then $\exists \sigma'(g') = \sigma^{-1}(o')(g)$ such that $\sigma(\sigma^{-1}(o')(g)) = o'(\sigma(g)) = o'(e)$ and $\sigma \circ \sigma'(g') = o'(e)$ so g' has the same *shape* as e.

Similarly, $\mathcal{H}$ satisfies the following property: for any state h in $\mathcal{H}$ and g in $\mathcal{G}'$ ($\mathcal{G}'$ is connected to $\mathcal{H}$ by one action through the substitution σ), all accessible substitutions from $\mathcal{G}'$ are also accessible from $\mathcal{H}$. $\mathcal{H}$ is the transitive closure of $\mathcal{G}$ in terms of substitutions. It is also possible to recursively prove that ShaPer finds a plan if there exists one. Given $s \in \mathcal{H}$ and a plan $P = a_n \circ \ldots \circ a_1$ such that $goal = P(s)$. If $a_1(s) \notin \mathcal{H}$ then there exists $\sigma(h) = a_1(s)$ with $a_1(s) \in \mathcal{G}'$ and $h \in \mathcal{H}$. Given i and the state $e = a_{i-1} \circ \ldots \circ a_1(s)$ and $e' = a_i(e)$ with $e \in \mathcal{G}'$ and $e' \notin \mathcal{G}'$. So $e' \in \mathcal{G}''$ (connected to $\mathcal{G}'$ by the state $g' \in \mathcal{G}'$ and a substitution θ: $\theta(g') = e'$). Owing to $\mathcal{H}$'s property, there exists a state h' of $\mathcal{H}$ and a substitution τ such that $\tau(g') = e'$. That means ShaPer is able to find a plan between s and e' (connect $\mathcal{H}$ to $\mathcal{G}''$ with one action); apply recursively this reasoning[7] on $a_n \circ \ldots \circ a_i$, demonstrates the completeness.

[6] Indeed $\sigma(o(s)) = \sigma((s - D_o) \cup A_o) = (\sigma(s) - \sigma(D_o)) \cup \sigma(A_o) = (\sigma(o))(\sigma(s))$

[7] If the plan P uses a third *shape* graph $\mathcal{G}^3$ (one-action-connected to $\mathcal{G}''$ which is also one-action-connected to $\mathcal{H}$) , the previous reasoning allows to prove that there exists a connection with one action between $\mathcal{H}$ and $\mathcal{G}^3$.

5 Results

In this section, we compare our planner with the most efficient planners from AIPS-2000 competition. FF [Hoffmann, 2000] and HSP [Bonet and Geffner, 1999] use heuristics based on a relaxed problem (plan without *Dellist*). IPP [Koehler *et al.*, 1997] and STAN [Fox and Long, 1999] are derived from GraphPlan [Blum and Furst, 1997]. In addition, they use an *on-line* mechanism that allows to deal with some symmetries; that is reason why it is interesting to compare them with ShaPer which is able to deal with a larger class of symmetries.

In this comparison, we do not mention TalPlan [Doherty and Kvarnstrom, 1999] because it is a domain dependent planner; the comparison can then be only made with ShaPer solution extraction step. In such a case, TalPlan exhibits clearly better results than ShaPer: the made-by-hand heuristics is still the best.

All running time presented in table 4 are measured on a Sparc Ultra 5 with 128Mb. '-' means that the planner does not find any solution in 3600sec. and '*' means that the *shape* graph has already been computed for a previous problem (same *domain subclass*).

The ferry and the gripper domains are interesting because they overwhelm the majority of the planners by the number of their applicable actions. Despite their ability to treat state symmetries, IPP and STAN can not solve large problems in both ferry and gripper domain. Ferry and gripper domains are easy for ShaPer because of their low number of *shapes* (resp. $2n + 1$ for n cars and $3n$ for n balls). Thanks to their good heuristics, HSP and FF solve easily all these problems (however, note that HSP solves gripper problems with an inefficient number of steps).

Blocks-world domains give surprising results for HSP and FF. HSP solves '-3' and '-2' problems more easily than '-1' problems. For FF, the situated is opposite. Problems labeled '-i' correspond to i-stacks problems: a goal for a '-1' problem is to put the first block under the stack (without changing the order of the other blocks); for '-2' and '-3' problems, the goal is to build one "interleaved" stack composed of all the blocks from the initial stacks (e.g. move stacks $s_1, s_2 \ldots$ and $p_1, p_2 \ldots$ into $s_1, p_1, s_2, p_2 \ldots$).

With several problems in the same *domain subclass*, *e.g.* all problems with 15 blocks, we better understand the importance of the *off-line* process (computed more efficiently than some HSP or FF solution): the graph building step is performed only once. The small number of *shapes* allows to have a very short computation time upper limit.

In conclusion, ShaPer solves all problems in less than 1 second; in addition it builds the graph and solves all problems faster than IPP and STAN (and often HSP) and extracts all problems faster than FF with a near-optimal number of plan steps.

6 Conclusion

We have proposed an original approach to task planning that allows to deal efficiently with large problems. It has been

	IPP-v4		STAN-v3		HSP-v2		FF-v2.2		ShaPer					
									off-line		*on-line*		*max*	
problem	step	time	step	time	step	time	step	time	nodes	time	step	time	step	time
ferry														
10 cars	39	3.00	39	7.2	39	0.08	39	0.04	21	0.02	39	0.01	41	0.01
20 cars	-	-	-	-	79	0.20	79	0.07	41	0.21	79	0.01	81	0.01
50 cars	-	-	-	-	199	5.50	199	0.20	101	4.86	199	0.07	201	0.08
gripper														
10 balls	29	56.9	29	202	37	0.10	29	0.03	30	0.07	29	0.01	55	0.01
20 balls	-	-	-	-	77	0.70	59	0.08	60	0.55	59	0.01	115	0.01
50 balls	-	-	-	-	197	18.80	149	0.30	150	14.04	149	0.15	295	0.27
blocks-world														
9-1 blocks	16	3.71	16	18.4	16	1.32	16	0.06	30	0.15	16	0.01	19	0.02
9-2 blocks	15	3.70	18	3.6	15	0.60	15	0.06	*	*	18	0.01	*	*
9-3 blocks	17	2.47	19	4.7	14	0.64	26	2.21	*	*	17	0.01	*	*
12-1 blocks	-	-	-	-	22	26.33	22	0.13	77	1.58	22	0.01	29	0.03
12-2 blocks	-	-	-	-	21	3.63	21	0.12	*	*	25	0.01	*	*
12-3 blocks	-	-	-	-	20	3.54	29	21.27	*	*	25	0.01	*	*
15-1 blocks	-	-	-	-	28	657.6	28	0.29	176	16.73	28	0.06	39	0.13
15-2 blocks	-	-	-	-	27	22.26	27	0.27	*	*	29	0.07	*	*
15-3 blocks	-	-	-	-	23	13.43	-	-	*	*	33	0.09	*	*
20-1 blocks	-	-	-	-	-	-	38	0.85	627	173.79	38	0.51	55	0.90
20-2 blocks	-	-	-	-	37	165.6	37	0.72	*	*	37	0.46	*	*
20-3 blocks	-	-	-	-	-	-	-	-	*	*	38	0.77	*	*

Table 4: Running time and quality (in number of Plan action) for some of the current best planners (see the AIPS-2000 competition); Comparing to ShaPer including its *off-line* process.

implemented in a domain independent task planner, called ShaPer. It performs in two steps. The first step is performed *off-line* and only once for a given *domain subclass*. It allows ShaPer to "capture the structure" of the state space and to store it in a data structure called the *shape* Graph. The main contribution here is the ability of ShaPer to build a very compact description when compared to the size of the complete state space. The *shape* graph is then used *on-line* by ShaPer to answer very efficiently to planning requests.

ShaPer exhibits several interesting properties: 1) it is complete, 2) It is possible to determine, after the *shape* Graph construction, the upper limit for the *on-line* solution extraction computation time as well as length of the longest plan, 3) it produces "good" (near-optimal) solutions.

Our future work will be devoted to a state decomposition method. Indeed, we would like to decompose a state into several disjoint parts in order to minimize interferences in action applicability. This should allow to generate sub-graphs and to deal with *sub-shapes* resulting in an even more compact state space description.

This would help to re-use *shape* graph created for a given *domain subclass* (*e.g.* 10 blocks) in order to generate *shapes* for "bigger" *subclasses* (*e.g.* 15 blocks) Besides, we hope to be able to exploit the structure of the *shape* graph for conformant and contingent planning as in [Guéré and Alami, 1999].

References

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

[Bonet and Geffner, 1999] B. Bonet and H. Geffner. Planning as heuristic search: New results. *5th European Conference on Planning (ECP'99)*, 1999.

[Doherty and Kvarnstrom, 1999] P. Doherty and J. Kvarnstrom. Talplanner: An empirical investigation of temporal logic-based forward chaining planner. In *Proc. 6th Int. Workshop on Temporal Representation and Reasoning*, 1999.

[Fikes and Nilsson, 1971] 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.

[Fox and Long, 1998] M. Fox and D. Long. The automatic inference of state invariants in tim. *AI Research (JAIR)*, 9:367–421, 1998.

[Fox and Long, 1999] 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.

[Gerevini and Schubert, 1998] A. Gerevini and L.K. Schubert. Inferring state constraints for domain-independent planning. In *Proc. 15th Nat. Conf. AI.(AAAI'98)*, 1998.

[Guéré and Alami, 1999] 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.

[Hoffmann, 2000] J Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *Proc. 12th Int. Symposium on Methodologies for Intelligent Systems*, 2000.

[Koehler *et al.*, 1997] 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.