Extraire et utiliser la structure des domaines en planification de tâches

Emmanuel Guere, Rachid Alami

Extraire et utiliser la structure des domaines en planification de tâches

Extract and use the domain's structure in task planning

Emmanuel Guéré[1] Rachid Alami[1]

[1] LAAS-CNRS
7, avenue du colonel Roche
31077 Toulouse cedex 4
France
{guere, rachid}@laas.fr

Résumé

Dans la course actuelle au planificateur de tâches le plus efficace, nous proposons dans cet article une nouvelle approche qui permet de traiter efficacement des problèmes de taille importante.
*L'algorithme se décompose en deux parties distinctes. Dans un premier temps*ShaPer *construit,* hors-ligne, *le graphe d'accessibilité, compact grâce à la notion de* forme, *d'un domaine donné. La capacité de ShaPer a «résister» à l'explosion combinatoire réside en effet dans une gestion efficace des symétries des domaines. Le graphe des* formes *est ensuite utilisé* en-ligne *pour répondre à n'importe quel problème de planification sur le domaine développé.*
Après avoir décrit en détail cette méthode originale, nous la complétons par l'utilisation et la synthèse automatique de hiérarchies d'abstractions, étendant ainsi les possibilités de ShaPer.
Une première implémentation a permis d'obtenir des résultats prometteurs sur la majorité des benchmarks, *rivalisant avec les meilleurs planificateurs de la compétition AIPS-2000.*

Mots Clef

Planification de tâches.

Abstract

In the stream of research that aims to speed up task planners, we propose a new approach to efficiently deal with large problems.
The algorithm performs in two steps. In the first step, executed off-line *for a given domain, ShaPer builds a compact representation of the accessibility state space owing to the* shape *notion. The main contribution of ShaPer is its ability to "resist" to combinatorial explosion thanks to the manipulation of sets of similar state descriptions. The* shape *graph is then used by ShaPer to answer very efficiently to planning requests on the learned domain.*
After detailing this original method, we extend it by introducing hierarchical abstractions (automatically built), in order to increase ShaPer's capabilities.
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 the AIPS-2000 competition.

Keywords

Task Planning.

1 Motivations

Un problème de planification consiste à élaborer une séquence d'actions (plan) permettant d'atteindre un état but défini depuis un état initial. Depuis quelques années plusieurs nouveaux algorithmes de planification rendent possible la résolution de problèmes de plus en plus complexes. Parmi ces planificateurs nous pouvons citer GraphPlan [1] et HSP [2] ainsi que leurs descendants (IPP [11], STAN [6], FF [9] ...). La recherche d'une solution se confond la plupart du temps avec une recherche dans l'espace d'état. Pour faire face à l'étendue des choix possibles, tous cherchent à élaguer certaines portions de l'espace de recherche non pertinentes pour le problème considéré.
Ainsi les dérivés de GraphPlan construisent, en temps polynômial par propagation de *mutex*, une approximation du graphe d'accessibilité. Ce graphe permet par exemple de savoir si deux propositions peuvent être vraies en même temps ; l'utilisation d'un tel graphe évite alors des recherches inutiles lors de l'extraction de solution en chaînage arrière.
HSP et FF, quant à eux, proposent une heuristique efficace basée sur une relaxation du problème de plani-

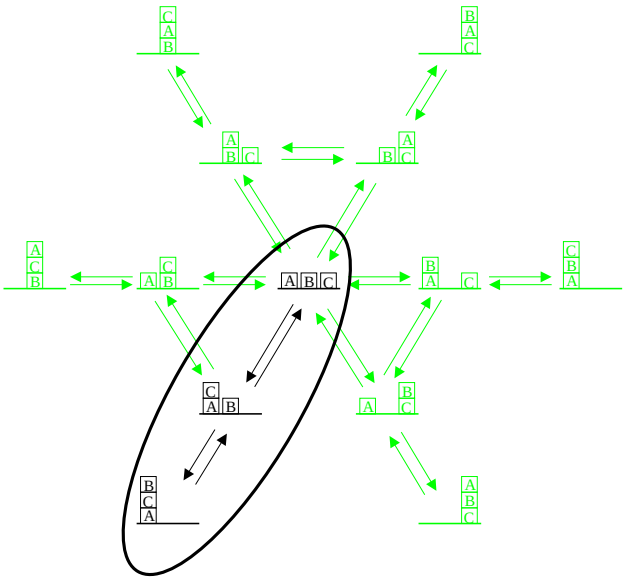


FIG. 1 – *Redondances de l'espace d'état.*

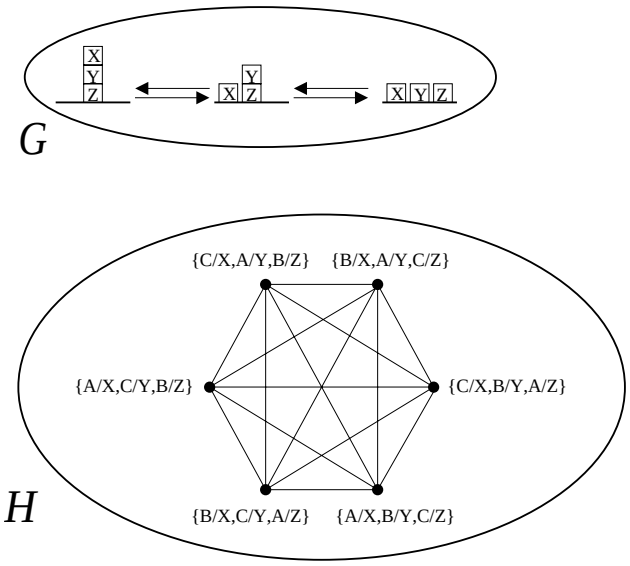


FIG. 2 – *L'espace d'état décrit par le graphe des* formes $\mathcal{G}$ *et le graphe des substitutions* $\mathcal{H}$.

fication en ne tenant plus compte des listes de retraits des opérateurs [1]. La distance séparant l'état courant du but est alors évaluée en fonction de la solution trouvée au problème relaxé. L'heuristique guide ainsi le planificateur en concentrant la recherche sur des portions pertinentes de l'espace d'état.

Un autre moyen de réduire l'espace de recherche consiste à éliminer les redondances. C'est ce que STAN [6] vise à réaliser lorsqu'il détecte les symétries du problème. Le calcul des symétries, effectué à partir d'une comparaison entre l'état initial et le but, permet de ne pas répéter inutilement les recherches sur des branches symétriques déjà explorées. De même il est possible d'analyser la structure du domaine de planification, par la recherche d'invariants (par exemple [7]), la détection automatique de types et la décomposition de problèmes [5, 12]. Il faut aussi mentionner les travaux sur la décomposition hiérarchique qui vise à diviser les prédicats en classes d'abstractions [13, 10].

La méthode que nous présentons dans cet article a pour but de compacter au maximum l'espace de recherche en ne conservant que les états qui pourront être utiles. La suite de cet article est illustrée par de nombreux exemples tous issus du monde des cubes.

Afin de prévenir une mauvaise interprétation, signalons que l'algorithme est valide quelque soit le domaine STRIPS utilisé, les cubes ne sont là que pour simplifier les dessins.

Le premier exemple que nous prenons est celui des 3-cubes. L'espace d'état correspondant à un problème où interviennent trois cubes est représenté sur la figure 1. Cet espace contient 13 états distincts ; cependant une certaine redondance dans ces états peut être remarquée. En effet tous les états ressemblent à un des trois états entourés, à une substitution près. Ceci remarqué, l'espace d'état peut être décrit de façon plus compacte par deux graphes : un premier graphe $\mathcal{G}$ que nous nommerons par la suite *graphe des formes* [2] et un second graphe $\mathcal{H}$ correspondant aux substitutions possibles (voir la figure 2).

Cet article propose une méthode originale de planification de tâches indépendante du domaine basée sur cette décomposition entre $\mathcal{G}$ et $\mathcal{H}$.

Le planificateur ainsi créé, nommé ShaPer [3], est capable de résoudre des problèmes importants rivalisant ainsi avec les meilleurs planificateurs de la compétition AIPS-2000. Outre ces résultats encourageants, ShaPer possède quelques propriétés remarquables. En effet le calcul du graphe des *formes* $\mathcal{G}$ n'est réalisé qu'une seule fois pour une classe de problèmes donnée et ce *hors-ligne*. La recherche *en-ligne* d'une solution à un problème donné se résumant à l'exploration de plusieurs $\mathcal{G}$, de taille connue, il est possible de borner le temps de recherche. Par exemple, ShaPer résoud n'importe quel problème de 20-cubes en moins d'une seconde. De plus comme le graphe des *formes* est ex-

1. Dans cet article, nous limitons l'exposé à une modélisation de type STRIPS [4]. Un opérateur STRIPS est composé de trois parties: une première nommée *préconditions*, P_α, qui indique si l'action α est applicable ou non à un état donné; une seconde, la *liste de retrait*, D_α, décrivant quelles propositions ne sont plus vraies après application de l'action et enfin la *liste d'ajout*, A_α, qui modélise les propositions nouvellement vraies. Ainsi l'action α n'est applicable à l'état S que si $P_\alpha \subseteq S$ et son application engendre l'état $(S - D_\alpha) \cup A_\alpha$.

2. Afin de simplifier, nous dirons dans un premier temps que deux états S_1 et S_2 sont de même *forme* s'il existe une substitution σ de variables telle que $\sigma(S_1) = S_2$.

3. *Sha*pe-based *P*lann*er*

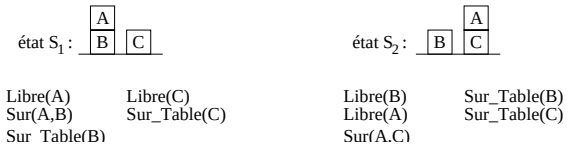
haustif, *i.e.* tout état accessible sera de même *forme* qu'un état de $\mathcal{G}$, il est possible d'en déduire des propriétés hiérarchiques pour le domaine considéré.
Avant de poursuivre plus en avant, nous allons définir la notion de *forme*, et décrire l'algorithme de construction de $\mathcal{G}$. Ceci fait, nous évoquerons la recherche de solution ainsi que les problèmes de complétude qui y sont liés. Une extension de l'approche par l'extraction et l'utilisation de hiérarchie nous permettra enfin de résoudre des problèmes de taille bien plus importante.

2 Comment trouver les *formes* ?

Comme le montrent les figures 1 et 2, le but du graphe $\mathcal{G}$ est de rassembler l'ensemble des *formes* accessibles depuis un état de départ S_{debut}, l'expansion du graphe $\mathcal{G}$ se faisant alors de façon incrémentale sur les états pertinents. La pertinence d'un état est évaluée en fonction de la nouveauté de sa *forme*. Une première définition des *formes* a été proposée dans [8]. Nous allons l'étendre dans cet article afin de couvrir un ensemble plus large de redondances.

2.1 Qu'est ce qu'une *forme* ?

Dans un souci d'élimination des redondances, deux états S_1 et S_2 sont dits de même *forme* si toutes les macro-actions (non-instanciées) applicables à S_1 le sont aussi à S_2 et inversement. Les macro-actions en question correspondent à n'importe quelle séquence d'actions STRIPS non-instanciées comme par exemple *Déplacer(X,Y,Z)* suivi de *Prendre(Y,X)*. Ainsi les deux états S_1 et S_2 sont de même *forme* [4].



La description des états sous forme propositionnelle de la figure précédente, montre clairement qu'il existe une substitution $\sigma = \{A/A, C/B, B/C\}$ entre S_1 et S_2. D'une façon générale, il est possible de montrer que s'il existe une substitution de variables σ telle que $\sigma(S_1) = S_2$ alors toute macro-action Γ applicable à S_1 est applicable à S_2 [5].
Avant de développer l'algorithme d'expansion de $\mathcal{G}$, remarquons que notre définition de *forme* coïncide bien avec notre volonté de réduction : si deux états ont les mêmes actions applicables, alors il n'apparaît pas utile d'explorer de nouveau les même actions.

4. Afin de prévenir toute confusion, les états S_1 et S_2 sont de même *forme* non pas parce que l'utilisateur a donné une description des piles, mais bien parce que les mêmes macro-actions sont applicables.

5. Soit $\Gamma(\chi)$ une instanciation de Γ applicable à S_1, *i.e.* $P_{\Gamma(\chi)} \subseteq S_1$; alors on peut démontrer que $P_{\Gamma(\sigma(\chi))} \subseteq S_2$.

2.2 $\mathcal{G}$ le graphe de toutes les *formes*

Pour construire le graphe des *formes* $\mathcal{G}$, nous allons développer toutes les actions applicables aux états dont la *forme* est nouvelle obtenant ainsi un graphe d'accessibilité.

Construire $\mathcal{G}$ une fois pour toutes ?

Supposons que notre graphe $\mathcal{G}$ contienne toutes les *formes* accessibles depuis l'état S_{debut} ; supposons maintenant que la *forme* de l'état S est incluse dans $\mathcal{G}$, on peut en déduire que toutes les *formes* accessibles depuis S sont contenues dans $\mathcal{G}$. Donc un même graphe $\mathcal{G}$ peut servir pour résoudre n'importe quel problème dont la *forme* de l'état initial s'y trouve.
En fait, pour une sous-classe de domaine donnée, il suffit de construire un seul graphe des *formes* par connexité. Ainsi si l'utilisateur donne l'ensemble des états des différentes connexités, ShaPer pourra répondre à n'importe quel problème de planification sans construire de nouveaux $\mathcal{G}$. En ce sens, nous pouvons dire que l'expansion du graphe des *formes* se fait *hors-ligne* alors que la recherche de solution à un problème précis se fait *en-ligne*.

Un algorithme en largeur d'abord

Supposons que l'on ait une fonction *Garder_l'état?*$(S,\mathcal{G})$ indiquant la pertinence de l'état S pour $\mathcal{G}$, *i.e.* la *forme* de S est nouvelle. L'algorithme 1 parcourt alors en largeur l'ensemble des états accessibles depuis S_{debut} jusqu'à obtention d'un point fixe.
Pour simplifier le propos, nous nous restreignons dans cet article à l'étude des domaines réversibles [6]. Il n'existe pas de différences majeures dans le traitement des domaines irréversibles, il faut juste s'assurer durant la

6. Pour chaque action $o \in \mathcal{O}$ applicable à un état S, il existe une contraposée $o^{-1} \in \mathcal{O}$ telle $o^{-1}(o(S)) = S$.

Alg. 1 CONSTRUIRE_$\mathcal{G}(S_{debut},\ \mathcal{O})$

```
◇ Génère toutes les formes accessibles depuis S_debut
Début
    G ← φ
    ND ← {S_debut}
    Tant Que ND ≠ φ Faire
        S ← Extraire(ND)
        Si Garder_l'état?(S, G) Alors
            Ajouter S à G
            Pour Tout o ∈ O tel que P_o ⊆ S Faire
                ND ← ND ∪ {o(S)}
            Fin Pour
        Sinon
            Si S ∈ G Alors
                Ajouter un arc à G
            Fin Si
        Fin Si
    Fin Tant Que
Fin
```

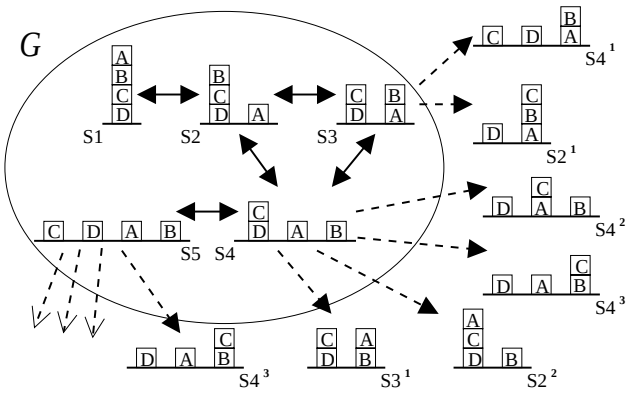


FIG. 3 – *Exemple de construction du graphe $\mathcal{G}$ avec 4-cubes.*

construction de $\mathcal{G}$ et l'extraction de solution que toutes les *formes* accessibles entre elles sont bien stockées. Par exemple si $g \in \mathcal{G}$ et g' sont de même *forme* et que $S \in \mathcal{G}$ accède à g', il faut s'assurer avant d'éliminer g' que g est accessible depuis S.

2.3 Un exemple de construction

Prenons l'exemple simple du domaine des 4-cubes en partant d'une seule pile (état S_1 de la figure 3). Une seule action est applicable à S_1 (poser le cube A sur la table), et son application engendre l'état S_2. L'état S_2 est de *forme* nouvelle, il est donc ajouté à $\mathcal{G}$ qui contient maintenant S_1 et S_2. Trois actions sont applicables à S_2, une première qui permet de revenir en S_1 (on ajoute donc cette nouvelle transition), une seconde qui pose le cube B sur le cube A pour engendrer l'état S_3 et enfin une troisième qui pose B sur la table donnant S_4. S_3 et S_4 représentant deux nouvelles *formes* et sont donc ajoutés à $\mathcal{G}$. L'algorithme développe alors l'état S_3. Quatre actions lui sont applicables donnant S_2, S_4, S_4^1 et S_2^1. L'état S_4^1 (resp. S_2^1) est de même *forme* que S_4 (resp. S_2) et n'est donc pas ajouté à $\mathcal{G}$. En parcourant ainsi en largeur la totalité des états de $\mathcal{G}$ nous obtenons le graphe des *formes* entouré sur la figure 3.

problème	Taille l'espace d'état	Nombre de noeuds de $\mathcal{G}$
Monde des cubes		
3 cubes	13	3
4 cubes	73	5
5 cubes	501	7
6 cubes	4051	11
7 cubes	41413	15
Domaine du gripper		
3 balles	88	9
4 balles	256	12
5 balles	704	15
6 balles	1856	18

TAB. 1 – *Évolution de la taille de l'espace d'état par rapport au graphe des* formes.

Le tableau 1 montre que notre méthode réduit dans des proportions exponentielles la taille de l'espace à considérer.

2.4 Comparer les *formes*

Lorsque que nous assimilons les *formes* aux substitutions (voir [8]), le test de pertinence d'un nouvel état S semble plus simple. En effet il suffit de parcourir tous les états g de $\mathcal{G}$ à la recherche d'une substitution σ telle que $\sigma(g) = S$. Cependant vérifier l'existence d'une substitution est un problème NP-Complet en théorie [7]. De plus notre notion de *forme* ayant évolué, il faut dès lors tester l'applicabilité de l'ensemble des macro-actions.

L'égalité de *forme* repose sur l'applicabilité d'un ensemble de séquences d'actions. Il est donc possible d'extraire de ces ensembles les actions pertinentes qui discriminent les états les uns des autres. Ainsi en parallèle de la construction du graphe $\mathcal{G}$, un arbre $\mathcal{F}$ de sélection de *forme* est synthétisé. Son but est double : guider la recherche d'égalité de *forme* de S dans $\mathcal{G}$ et entamer la construction d'une substitution σ qui assurera l'égalité des *formes*.

Rappelons en effet que s'il existe une substitution entre deux états, nous avons alors la certitude que ces deux états sont de même *forme*.

L'algorithme 2 se décompose en deux parties : une première qui se charge, grâce à l'arbre $\mathcal{F}$, de trouver le seul candidat possible g de $\mathcal{G}$ à une *forme* identique à celle de S ; tandis que la seconde doit quant à elle valider ces présomptions. Deux solutions sont alors envisageables : soit il existe une substitution (le parcours de l'arbre a permis de la construire partiellement) auquel cas il ne faut pas garder l'état, soit il n'existe pas de substitution et il faut alors trouver une action qui discrimine g de S. Si g et S sont de même *forme*, une telle action n'existe pas, il faut alors garantir que les propositions des états qui n'ont pas été utilisées ne permettront jamais de déclencher une action.

La recherche de cette séquence d'actions jouant le rôle de contre-exemple peut être efficacement guidée par l'heuristique suivante :

- $P_\Gamma \cap (g - S) \neq \phi$ où Γ représente la séquence recherchée. En effet, si aucune précondition de Γ ne se trouve dans $g - S$ la séquence sera applicable à la fois à g et à S ;
- $P_\Gamma - P_\Upsilon \neq \phi$ où Υ symbolise la séquence d'action communes de l'arbre $\mathcal{F}$. Cela revient à dire qu'il est inutile de tester deux fois les mêmes préconditions ;

7. Cela revient à tester l'isomorphisme de deux graphes.

Alg. 2 GARDER_L'ÉTAT?(S, $\mathcal{G}$, $\mathcal{F}$, Δ): BOOLÉEN

◇ *Teste l'appartenance de S à l'arbre $\mathcal{F}$*
◇ *et évalue ensuite si c'est une nouvelle* forme *ou non.*
Début
  $f \leftarrow \mathcal{F}$
  $\Upsilon \leftarrow \phi$ ◇ *Séquence d'action applicable à S*
  Tant Que f n'est pas une feuille **Faire**
    Si f.action est applicable à S **Alors**
      f ← f.suite_action
      Composer Υ et f.action
    Sinon
      f ← f.suite_pas_action
    Fin Si
  Fin Tant Que
  Soit $g \in \mathcal{G}$ l'état feuille de f trouvé précédemment
  Si $\sigma(g) = S$ **Alors**
    Retour Faux
  Sinon
    Chercher Γ une macro-action
      ◇ *applicable à g et pas à S (ou inversement)*
    Si trouver Γ **Alors**
      f.action ← Γ
      f.suite_action ← g ou S selon
      f.suite_pas_action ← S ou g selon
      Retour Vrai
    Sinon
      Retour Faux
  Fin Si
Fin

- $P_\Gamma \cap P_\Upsilon \neq \phi$ correspond à une heuristique plus empirique qui vise à maximiser les contraintes entre arguments afin de linéariser au maximum le choix des substitutions possibles.

2.5 Un exemple de construction avec l'arbre

La construction de l'arbre $\mathcal{F}$, comme l'explicite la figure 4, s'effectue donc de manière incrémentale. Nous considérons qu'une séquence d'actions est pertinente pour l'arbre s'il n'existe pas de séquence plus courte discriminant les mêmes états.
Afin de mieux comprendre le rôle et la construction de cet arbre prenons l'exemple du monde des 4-cubes. Nous commençons la construction de $\mathcal{G}$ et $\mathcal{F}$ avec l'état de début S_1 de la figure 3. Décrivons la construction de $\mathcal{F}$ en parallèle de $\mathcal{G}$ (voir figure 4) :

-a- Le graphe $\mathcal{G}$ est initialisé avec S_1. Pour l'instant tous les noeuds de $\mathcal{G}$ sont de même *forme*, $\mathcal{F}$ exprime cela par un seul noeud de label S_1 ;

-b- L'expansion de $\mathcal{G}$ génère l'état S_2. Cet état n'est pas de même *forme* que S_1. En effet l'action *Déplacer(X,Y,Z)* est applicable à S_2 (si l'on considère l'instanciation $\{B/X,\ C/Y,\ A/Z\}$) mais pas à S_1, l'arbre $\mathcal{F}$ est donc modifié par l'introduction

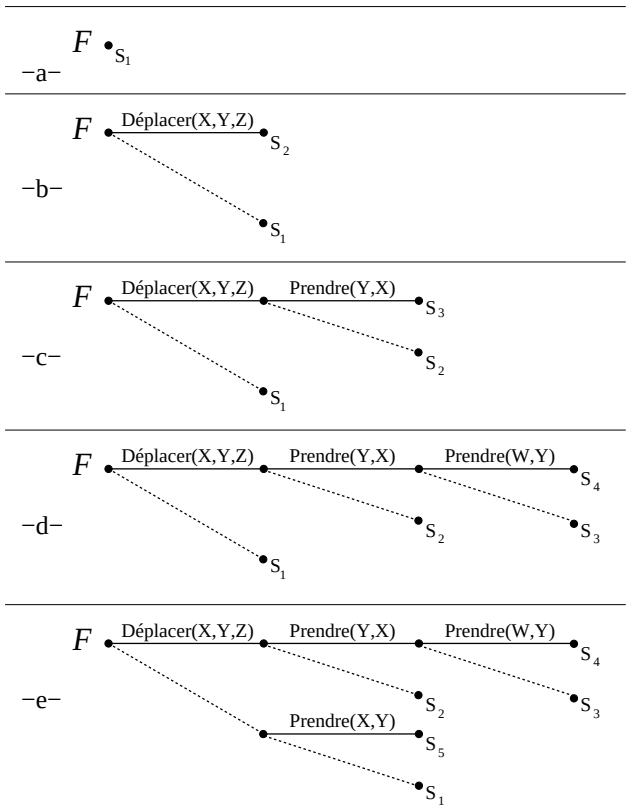


FIG. 4 – *Construction de l'arbre de sélection des* formes $\mathcal{F}$.

de S_2. Le trait en pointillé signifie que l'action *Déplacer(X,Y,Z)* n'est pas applicable à S_1. L'arbre peut alors se comprendre par : si l'expansion du graphe des *formes* $\mathcal{G}$ est finie, alors tous les états où *Déplacer(X,Y,Z)* est applicable sont de même *forme* que S_2 et sinon de même *forme* que S_1 ;

-c- L'action *Déplacer(X,Y,Z)* est applicable au nouvel état S_3 avec la substitution $\{C/X,\ D/Y,\ B/Z\}$. Deux cas se présentent alors : soit S_2 et S_3 sont de même *forme*, soit ils sont de *formes* différentes, auquel cas il existe une action α applicable à seulement un des deux états (on est néanmoins certain que S_3 n'est pas de même *forme* que S_1). Dans notre exemple l'action *Prendre(Y,X)* avec $\{D/X,\ C/Y\}$ est applicable à l'état résultant de l'application de *Déplacer(C,D,B)* à S_3 mais l'action *Prendre(C,B)* n'est pas applicable au résultat de *Déplacer(B,C,A)* sur S_2 ;

-d- De la même manière l'état S_4 peut se confondre dans $\mathcal{F}$ avec S_3 ; cette indéterminée est levée par l'introduction de l'action *Prendre(T,Y)* ;

-e- Enfin l'état S_5 se confond avec S_1 car *Déplacer(X,-Y,Z)* n'est pas applicable. L'action *Prendre(X,Y)* permet cependant de les différencier.

2.6 Améliorations dues à $\mathcal{F}$

La construction de l'arbre $\mathcal{F}$ permet donc en un seul parcours et une seule recherche de substitution de savoir si oui ou non la *forme* d'un état est nouvelle pour $\mathcal{G}$. Le tableau 2 propose une comparaison, sur le domaine des cubes, entre les temps de génération de $\mathcal{G}$ avec ou sans l'aide de $\mathcal{F}$. Rappelons (voir [8]) que le test de la nouveauté d'une *forme* avant l'introduction de $\mathcal{F}$, nécessitait une recherche exhaustive, *i.e.* sur tous les états de $\mathcal{G}$, de substitutions. $|\mathcal{E}|$ correspond au nombre d'états développés, nous pouvons noter que ce nombre est borné par $|\mathcal{G}| \cdot N$ où N représente le nombre maximum d'actions applicables à un état.

3 Recherche d'un plan solution

Le graphe $\mathcal{G}$ étant construit, il ne reste plus qu'à développer le graphe des substitutions $\mathcal{H}$. Comme notre but est d'utiliser ces graphes pour résoudre n'importe quel problème de planification de la sous-classe de domaine correspondante, il n'est pas nécessaire de construire tout $\mathcal{H}$.

L'idée est la suivante : si l'état initial S_{init} est de même forme qu'un état $g \in \mathcal{G}$, alors il est possible d'instancier le graphe des *formes* $\mathcal{G}_{init}$ qui contient toutes les *formes* accessibles depuis S_{init}. Dans ce cas, l'état but S_{but} sera forcément (s'il existe une solution) accessible *via* un certain nombre d'instanciations de $\mathcal{G}$. La question qui se pose alors est de savoir s'il est possible de construire un graphe des substitutions $\mathcal{H}$ minimal qui garantisse une solution *via* seulement des substitutions de $\mathcal{H}$. Le problème se résume finalement à trouver une action connectant $\mathcal{H}_{init}$ à $\mathcal{G}_{but}$.

3.1 Construire un $\mathcal{H}$ minimal

Pour assurer une connexion en une seule action, il faut que toutes les substitutions accessibles depuis l'état initial soient accessibles directement, *i.e.* en une seule action, depuis $\mathcal{H}$. Ainsi supposons que $\mathcal{G}'$ soit accessible directement depuis $\mathcal{H}$ et que $\mathcal{G}''$ le soit depuis $\mathcal{G}'$, pour garantir la complétude il faut que $\mathcal{G}''$ soit directement accessible depuis $\mathcal{H}$. Si ce n'est pas le cas, il faut ajouter la substitution qui engendre $\mathcal{G}'$ à $\mathcal{H}$.

Les résultats empiriques montrent que l'ensemble des substitutions pertinentes sont très rapidement trouvées car très souvent répétées. C'est pourquoi l'algorithme 3 propose une recherche stochastique plutôt qu'exhaustive.

Cubes	$\lvert\mathcal{G}\rvert$	$\lvert\mathcal{E}\rvert$	CPU $-\mathcal{F}$	CPU $+\ \mathcal{F}$
5	7	56	0.01	0.01
10	42	951	0.30	0.11
15	176	7186	16.73	1.47
20	627	38307	173.79	13.01
25	1958	163187	2223.76	84.55
30	5604	599292	28182.24	423.92

TAB. 2 – *Gain dû à l'utilisation de l'arbre $\mathcal{F}$.*

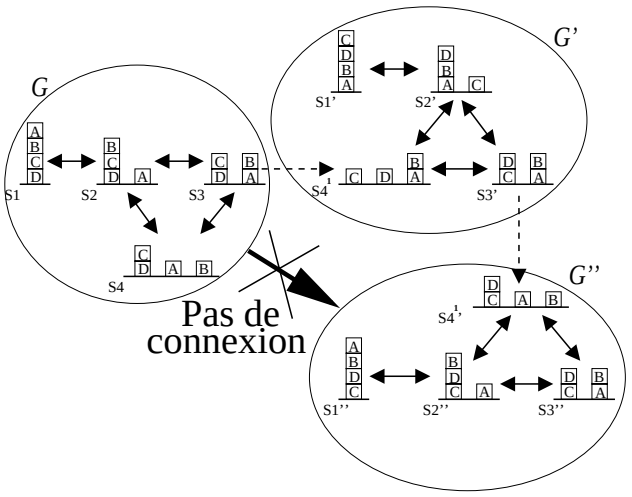


FIG. 5 – *$\mathcal{G}$ n'est pas suffisant pour assurer la complétude.*

Exemple de construction

Prenons le monde des 4-cubes comme exemple. Étant donné que l'état dans lequel les quatre cubes se trouvent sur la table permet d'accéder à toutes les substitutions, nous allons restreindre le domaine à trois piles. Le graphe des *formes* associé est représenté par $\mathcal{G}$ sur la figure 5. Cette figure montre clairement que le graphe $\mathcal{G}'$ issu de S'_4 permet d'atteindre un graphe $\mathcal{G}''$ non accessible directement depuis $\mathcal{G}$, il convient alors d'augmenter $\mathcal{H}$. Le graphe ainsi obtenu est minimal et ne comporte que deux substitutions distinctes comme le montre la figure 6 qui représente la réduction de l'espace d'état. Il est maintenant possible de résoudre

Alg. 3 CONSTRUIRE_$\mathcal{H}(S_{debut},\ \mathcal{G})$

◇ *Génère le graphe $\mathcal{H}$ minimal pour résoudre tous les*
◇ *problèmes en une seule action*

Début
  $\mathcal{H} \leftarrow Id$
  $Couverture \leftarrow 0$
  Tant Que $Couverture < Prob$ **Faire**
    Choisir aléatoirement un $\mathcal{G}'$ directement connecté à $\mathcal{H}$
    Choisir aléatoirement un $\mathcal{G}''$ directement connecté à $\mathcal{G}'$
    Si $\mathcal{G}''$ directement connecté à $\mathcal{H}$ **Alors**
      $Couverture \leftarrow Couverture + 1$
    Sinon
      $Couverture \leftarrow 0$
      Compléter $\mathcal{H}$ avec $\mathcal{G}'$
    Fin Si
  Fin Tant Que
Fin

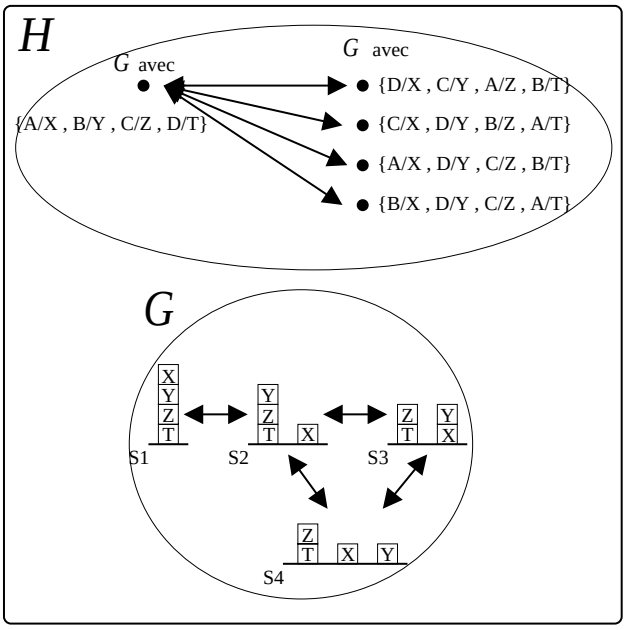


FIG. 6 – *L'espace d'état réduit représenté par $\mathcal{H}$ et $\mathcal{G}$.*

n'importe quel problème de 4-cubes limité à trois piles en connectant $\mathcal{H}_{init}$ et $\mathcal{G}_{but}$ *via* une seule action.

3.2 Une action suffit

Les graphes $\mathcal{G}$ et $\mathcal{H}$ construits *hors-ligne*, il suffit maintenant pour extraire une solution d'instancier les graphes avec respectivement l'état initial et le but, *i.e.* $\mathcal{H}_{init}$ et $\mathcal{G}_{but}$, puis de les relier. C'est ce que propose l'algorithme 4.

Alg. 4 EXTRAIRE_SOLUTION(S_{init}, S_{but}, $\mathcal{G}$, $\mathcal{H}$)

◇ *Parcourt tous les états de $\mathcal{H}_{init}$ à la recherche*
◇ *d'une connexion avec $\mathcal{G}_{but}$*

Début
  Si $\exists \sigma, h \in \mathcal{H}.\sigma(h) = S_{init}$ Alors
    Si $\exists \sigma', g \in \mathcal{G}.S_{but} \subseteq \sigma'(g)$ **Alors**
      Construire $\mathcal{H}_{init}$ et $\mathcal{G}_{but}$
      Pour Tout $h \in \mathcal{H}_{init}$ **Faire**
        Si $\exists o \in \mathcal{O}.o(h) \in \mathcal{G}_{but}$ **Alors**
          Retour $Plan(S_{init} \rightarrow h) + o + Plan(o(h) \rightarrow S_{but})$
        Fin Si
      Fin Pour
      Retour Pas de solution
    Sinon
      Retour Pas de solution
    Fin Si
  Sinon
    $\mathcal{G}' \leftarrow Construire\_\mathcal{G}(S_{init}, \mathcal{O})$
    Retour Extraire_Solution(S_{init}, S_{but}, $\mathcal{G}'$)
  Fin Si
Fin

La figure 7 décompose la recherche de solution pour un problème de 4-cubes limité à trois piles.

Recherche en temps borné

Cette méthode présente l'avantage de fournir une réponse en temps borné. En effet, puisque l'extraction de solution se résume à parcourir tous les états de $\mathcal{H}$, construit *hors-ligne* donc indépendant du problème, le nombre maximum d'états est connu à l'avance. Une extrapolation du temps nécessaire au traitement d'un état permet donc de borner le temps de recherche. Le nombre de tests à effectuer est au maximum $|\mathcal{E}|$ (voir le tableau 2). Ainsi par exemple, tous les problèmes de 20-cubes seront résolus en moins d'une seconde. En appliquant ce raisonnement, il apparaît clairement que la taille du plan est elle aussi bornée.

4 Utilisation et génération de hiérarchies

ShaPer permet de réduire efficacement l'espace de recherche grâce à sa gestion des redondances particulièrement efficace sur des domaines contenant beaucoup d'objets de type identique. Cependant, il est aussi vrai que la taille du graphe des *formes* $\mathcal{G}$ tend vers la taille de l'espace d'état lorsque le nombre d'objets de même type tend vers 1. Pour traiter convenablement des domaines dans lesquels plusieurs types d'objets sont indépendants les uns des autres, nous allons introduire une décomposition automatique en niveaux hiérarchiques.

4.1 Propriété de Monotonicité Ordonnée

La décomposition hiérarchique a pour but de diviser un domaine de planification en sous-domaines (classes d'abstraction) afin de faciliter la recherche de plan so-

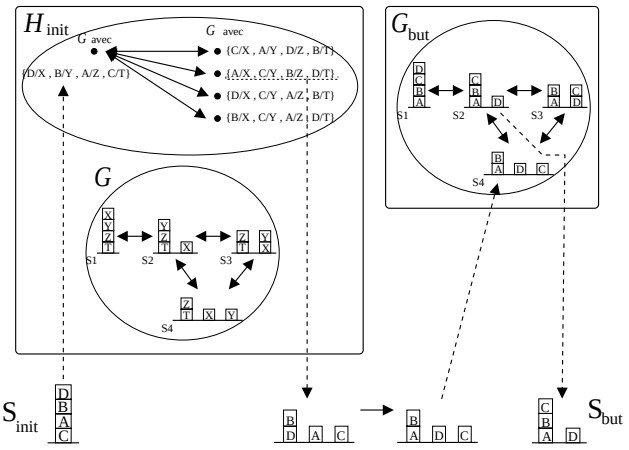


FIG. 7 – *Un exemple de solution pour les 4-cubes restreint sur 3-piles.*

lution. Pour cela, les différentes classes d'abstraction sont partiellement ordonnées entre elles. La propriété de monotonicité ordonnée introduite par [10], garantit que la résolution d'un niveau hiérarchique supérieur ne sera pas remise en cause ensuite par la résolution d'un niveau inférieur. Il suffit alors de résoudre les buts par ordre de niveau hiérarchique.

Pour ordonner automatiquement les prédicats entre eux (et par là même les actions), il suffit de vérifier les deux conditions suivantes : si deux prédicats apparaissent en effets d'une même action, alors ils sont de même niveau hiérarchique ; les effets ont un niveau hiérarchique supérieur ou égal aux préconditions.

Prenons l'exemple simple d'un robot disposant des trois actions *Prendre* un objet, le *Poser*, et se *Déplacer* définies par :

Prendre(X,Y)
- pre: Est_En(Y), Sur(X,Y), Tient_Rien
- ajout: Tient(X)
- retrait: Sur(X,Y), Tient_Rien

Poser(X,Y)
- pre: Est_En(Y), Tient(X)
- ajout: Sur(X,Y), Tient_Rien
- retrait: Tient(X)

Déplacer(X,Y)
- pre: Est_En(X), Connecté(X,Y)
- ajout: Est_En(Y)
- retrait: Est_En(X)

La décomposition hiérarchique de ce domaine donne l'ordre total suivant :

({Tient,Tient_Rien,Sur} ,3)
({Est_En} ,2)
({Connecté} ,1)

Ainsi pour résoudre un problème, le planificateur va d'abord chercher un plan pour les prédicats {Tient, Sur, Tient_Rien}, puis complétera le plan avec le prédicat Est_En sans avoir a remettre en cause le niveau supérieur. Il est important de noter que cette décomposition ne donne lieu à aucune approximation, *i.e.* le problème peut donc encore être résolu de façon optimale.

4.2 Adapter les hiérarchies à ShaPer

Ainsi il apparaît évident que le graphe $\mathcal{G}$ pour le prédicat Est_En ne sera pas remis en cause par les actions applicables aux prédicats {Tient,Tient_Rien,Sur}. Il est donc possible de construire un graphe de *forme* $\mathcal{G}_i$ par niveau hiérarchique n_i.

Pour cela il faut commencer par construire les niveaux inférieurs pour ensuite les utiliser comme préconditions des actions des niveaux supérieurs. L'exemple précédent engendrera alors deux graphes $\mathcal{G}_1$ et $\mathcal{G}_2$, le second utilisant les positions permises par le premier. L'utilisation des hiérarchies modifie légèrement l'extraction de solution. En effet il faut maintenant chercher une solution pour chaque niveau (comme s'il s'agissait d'un graphe normal), par ordre décroissant.

La recherche d'une solution se fait toujours en temps borné, mais il faut maintenant prendre en compte tous les niveaux hiérarchiques.

Evaluons cette borne sur seulement deux niveaux successifs n_{i-1} et n_i avec leur graphe respectif $\mathcal{G}_{i-1}$ et $\mathcal{G}_i$. Soit t_i le temps maximal nécessaire pour trouver une solution dans le graphe $\mathcal{G}_i$ et a_i la taille maximale d'un plan. Ainsi un plan solution du niveau i prend au plus t_i et utilise au plus a_i actions. Compte tenu de la hiérarchie, il peut s'avérer nécessaire de chercher un plan dans le niveau $i-1$ entre chaque action, le temps total de recherche étant alors borné par $t_i + a_i \cdot t_{i-1}$.

4.3 Extraire une décomposition hiérarchique

Dans l'état actuel, pour résoudre un problème de planification, ShaPer parcourt tous les états de $\mathcal{H}_{init}$ dans le but d'en connecter un avec $\mathcal{G}_{but}$. Cette partie a pour objectif d'utiliser la notion de hiérarchie entre proposition pour guider la recherche dans $\mathcal{H}_{init}$.

En effet, par définition de la méthode, une seule action suffit pour connecter $\mathcal{H}_{init}$ à $\mathcal{G}_{but}$; la difficulté réside donc dans la recherche d'un chemin au sein de $\mathcal{H}_{init}$. Comme l'état but se trouve dans $\mathcal{G}_{but}$, l'idée est d'utiliser la connaissance sur l'exhaustivité de ses *formes* pour essayer de s'en approcher le plus possible pendant le parcours de $\mathcal{H}_{init}$.

Par exemple si le fait p est présent dans les états de $\mathcal{G}_{but}$, alors une bonne heuristique pour parcourir $\mathcal{H}_{init}$ serait de chercher à obtenir p. C'est sur ce principe que nous utilisons la notion de hiérarchie pour ordonner les propositions de $\mathcal{G}$ entre elles.

Pour cela il suffit de considérer toutes les actions du graphe des *formes*. L'ensemble est alors soumis à l'algorithme de [10] pour en extraire un ordre partiel. A cause des cycles qui peuvent se présenter au sein du graphe, il se peut qu'on fusionne différentes classes d'abstraction entre elles. Pour éviter cette perte d'informaion, il faut conserver la hiérarchie la plus fine (en conservant la cohérence).

Afin de mieux comprendre, prenons l'exemple des 4-cubes de la figure 3. En suivant le chemin S_1 - S_2 - S_4 - S_5, nous obtenons l'ordre total :

({ Sur_Table(Z), Libre(T), Sur(Z,T)} ,4)
({ Sur_Table(Y), Libre(Z), Sur(Y,Z)} ,3)
({ Sur_Table(X), Libre(Y), Sur(X,Y)} ,2)
({Libre(X)} ,1)

Par contre en suivant le chemin S_1 - S_2 - S_3 - S_4 - S_5, l'ordre obtenu est :

({Sur_Table(Z), Libre(T),Sur(Z,T)} ,2)
({Sur_Table(Y), Libre(Z), Sur(Y,Z), Sur_Table(X), Libre(Y), Sur(X,Y), Libre(X)} ,1)

La première hiérarchie est conservée compte tenu du fait que la seconde est moins précise et non contradictoire.

Le parcours du graphe $\mathcal{H}$ peut donc maintenant être guidé avantageusement par cette hiérarchie. Par exemple dans le monde des cubes, tous les problèmes de moins de 30 cubes peuvent être résolus en moins de 0.1 seconde. Cependant il est important de noter que cette hiérarchie ne correspondant pas à une hiérarchie

du domaine, mais seulement à une heuristique, on perd encore un peu plus d'optimalité (car l'heuristique proposée n'est pas admissible).
L'algorithme 4 peut, une fois transformé, retourner la meilleure solution que notre méthode permette. Pour cela, il suffit de parcourir $\mathcal{H}_{init}$ et $\mathcal{G}_{but}$ par ordre croissant de $Plan(S_{init} \rightarrow h) + o + Plan(o(h) \rightarrow S_{but})$. C'est d'ailleurs de cette façon que les résultats présentés dans le tableau 3 ont été obtenus afin de trouver les plus courts plans possibles aux dépens du temps de calcul.
Cependant l'utilisation de l'heuristique permet d'atteindre une solution très rapidement et s'il reste du temps, il est toujours possible de continuer à explorer $\mathcal{H}_{init}$ pour trouver une meilleure connexion. En cela, une telle recherche de solution pourrait être considérée comme *any-time*.
Il est tout de même possible de générer une hiérarchie du domaine qui pourrait ensuite être utilisée par n'importe quel planificateur. Pour cela il faut prendre en compte toutes les substitutions possibles. Évidemment pour le monde des cubes cela revient à remettre à plat tous les prédicats. Par contre sur des domaines comme les tours de Hanoï, il est possible de générer une hiérarchie pertinente. Prenons le domaine des 3-disques décrit par la seule action [8] :

Déplacer(X,Y,Z)
pre: Libre(X), Sur(X,Y), Libre(Z), Inf(X,Z)
ajout: Sur(X,Z), Libre(Y)
retrait: Sur(X,Y), Libre(Z)

Aucune considération sur les prédicats ne permet de conclure à une quelconque hiérarchie. Néanmoins notre algorithme engendre la hiérarchie :
({Libre(R), Libre(C), Sur(B,C), Sur(B,R)} ,4)
({Libre(B), Libre(Q), Sur(A,B), Sur(A,Q)} ,3)
({Libre(A)},2)
({Inf(A,B), Inf(A,C) ...},1)
Une fois toutes les substitutions possibles prises en compte, quatre classes d'abstraction apparaissent :
({Libre(_),Libre(C), Sur(B,C), Sur(B,_)} ,4)
({Libre(B), Sur(A,B), Sur(A,_)} ,3)
({Libre(A)} ,2)
({Inf(A,B), Inf(A,C) ...},1)
où '_' représente un quelconque pique. Grâce à cette décomposition hiérarchique, il apparaît clairement qu'il vaut mieux poser B sur C avant de poser A sur B.

5 Résultats

Cette dernière partie compare les résultats expérimentaux de ShaPer à ceux des meilleurs planificateurs de la compétition AIPS-2000, à savoir IPP [11], STAN [6], HSP [2] et FF [9], sur le domaine du *ferry*, du *gripper* et des cubes. Le domaine *logistics* n'apparaît pas dans ce comparatif, parce qu'il se comporte exactement, une fois la décomposition hiérarchique du domaine effectuée, comme le domaine des cubes. En effet, abstraction faite des avions et camions, il ne reste plus que la position des paquets. Ceux-ci se regroupent sur chaque site à la façon des piles (les permutations en moins), le graphe des *formes* est donc le même que pour les cubes.
Notons aussi que dans ce comparatif, nous ne mentionnons pas TalPlan [3], puisque c'est un planificateur dépendant du domaine. Cependant, une fois le graphe $\mathcal{H}$ construit, notre planificateur pourrait lui aussi être considéré comme dépendant du domaine (de $\mathcal{H}$ en fait). La comparaison entre TalPlan et ShaPer s'effectuerait alors sur la recherche de solution. Dans ce cas TalPlan apparaîtrait clairement supérieur, l'heuristique donnée par l'utilisateur restant bien meilleure.
Les tests ont été effectués sur une station SUN-Ultra 5 avec 128Mo. La recherche de plan est stoppée au bout d'une heure (symbolisé par '-'), et le caractère '*' signifie que le graphe a déjà été construit ; il n'est donc pas nécessaire de recommencer. L'intérêt de la construction *hors-ligne* apparaît plus clairement au vu des résultats des problèmes '-2' et '-3' pour lesquels il n'est pas nécessaire de reconstruire le graphe.
Nous pouvons remarquer que malgré son habileté à regrouper les symétries, STAN ne réussit pas à résoudre plus de problèmes que IPP sur les domaines du *ferry* et du *gripper* qui sont pourtant intéressants pour leurs symétries. En effet, il n'existe que $2n+1$ *formes* pour n voitures et $3n$ *formes* pour n balles. Ceci explique les excellents résultats de ShaPer sur ces domaines où la construction du graphe reste polynomiale.
Afin de confondre un peu HSP et FF qui obtiennent d'excellents résultats dans de nombreux domaines, nous utilisons trois catégories de tests pour les cubes. Les problèmes notés '-i' sont définis de la manière suivante : l'état initial est composé de i piles et le but correspond a un enchevêtrement en une seule pile. Par exemple '-2' commence avec deux piles $b_1, b_2, \ldots$ et $c_1, c_2, \ldots$ et termine par $b_1, c_1, b_2, c_2, \ldots$
Il est surprenant de constater que FF et HSP ne se comportent pas de la même manière alors qu'ils ont des heuristiques assez proches (HSP résout plus facilement les enchevêtrements complexes contrairement à FF).
Notons sur ces exemples les temps de recherche au pire cas de ShaPer qui ne dépassent jamais la seconde (voir la dernière colonne qui présente le temps maximal de recherche et le nombre maximal d'actions du plan).
De plus, ShaPer est toujours plus efficace que HSP même en ajoutant le temps *hors-ligne*, et que FF en ne considérant que les temps de recherche.

6 Conclusion et perspectives

Cet article propose une approche originale à la planification de tâches indépendante du domaine, tout en présentant des améliorations tant au niveau de la construction de $\mathcal{G}$ que extraction de solution par rapport aux premiers résultats publiés dans [8].

8. Dans ce domaine, les piques $\{P,Q,R\}$ sont en fait considérés comme de très gros disques. L'ordre entre les disques est alors $A < B < C < \{P,Q,R\}$.

	IPP-v4		STAN-v3		HSP-v2		FF-v2.2		ShaPer					
									hors-ligne		*en-ligne*		*max*	
problème	$\|\Gamma\|$	CPU	$\|\Gamma\|$	CPU	$\|\Gamma\|$	CPU	$\|\Gamma\|$	CPU	états	CPU	$\|\Gamma\|$	CPU	$\|\Gamma\|$	CPU
Domaine du *ferry*														
10	39	3.00	39	7.2	39	0.08	39	0.04	21	0.02	39	0.01	41	0.01
20	-	-	-	-	79	0.20	79	0.07	41	0.13	79	0.01	81	0.01
50	-	-	-	-	199	5.50	199	0.20	101	2.63	199	0.07	201	0.08
Domaine du *gripper*														
10	29	56.9	29	202	37	0.10	29	0.03	30	0.02	29	0.01	55	0.01
20	-	-	-	-	77	0.70	59	0.08	60	0.22	59	0.01	115	0.01
50	-	-	-	-	197	18.80	149	0.30	150	8.11	149	0.15	295	0.27
Domaine des *cubes*														
9-1	16	3.71	16	18.4	16	1.32	16	0.06	30	0.10	16	0.01	19	0.02
9-2	15	3.70	18	3.6	15	0.60	15	0.06	*	*	18	0.01	*	*
9-3	17	2.47	19	4.7	14	0.64	26	2.21	*	*	17	0.01	*	*
12-1	-	-	-	-	22	26.33	22	0.13	77	1.06	22	0.01	29	0.03
12-2	-	-	-	-	21	3.63	21	0.12	*	*	25	0.01	*	*
12-3	-	-	-	-	20	3.54	29	21.27	*	*	25	0.01	*	*
15-1	-	-	-	-	28	657.6	28	0.29	176	1.47	28	0.06	39	0.13
15-2	-	-	-	-	27	22.26	27	0.27	*	*	29	0.07	*	*
15-3	-	-	-	-	23	13.43	-	-	*	*	33	0.09	*	*
20-1	-	-	-	-	-	-	38	0.85	627	13.01	38	0.51	55	0.90
20-2	-	-	-	-	37	165.6	37	0.72	*	*	37	0.46	*	*
20-3	-	-	-	-	-	-	-	-	*	*	38	0.77	*	*

TAB. 3 – *Comparatif entre les meilleurs planificateurs actuels (voir la compétition d'AIPS-2000) et ShaPer. Sont inclus les temps* hors-ligne *et les bornes.*

ShaPer peut, grâce à sa gestion efficace des redondances, résoudre des problèmes de grande envergure, rivalisant ainsi avec les meilleurs planificateurs actuels. L'algorithme se divise en deux parties distinctes ; une première qui se charge de développer un graphe d'accessibilité, et une seconde qui en extrait une solution pour le problème posé. Chacune des ces deux parties présente une propriété extrêmement utile pour traiter des problèmes réalistes.

Le calcul du graphe peut être effectué *hors-ligne* sans pour autant spécifier un problème précis ; la recherche d'une solution s'effectue alors *en-ligne* et en temps borné, propriété qui peut être très utile pour des applications réactives. Par exemple, dans le pire cas, la recherche d'une solution pour un quelconque problème de 20-cubes prendra une seconde, pire cas qui est rarement attient grâce à l'heuristique de recherche mise en place par ordonnancement des propositions.

A cela s'ajoute une décomposition hiérarchique automatique qui vient renforcer les capacités de ShaPer à résoudre des problèmes contenant de nombreux types d'objets indépendants.

Il reste cependant de nombreux travaux à effectuer. Notamment, le graphe des *formes* offre des perspectives intéressantes, en tant qu'ensemble de *mutex* généralisés, pour guider la recherche de planificateurs à *moindre engagement*. De plus, sa recherche en chaînage avant couplée à un ensemble d'actions applicables ($\mathcal{G}$), devrait grandement faciliter la résolution de problèmes dits de *conformant planning*.

Références

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

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

[6] 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.

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

[8] E. Guéré and R. Alami. One action is enough to plan. *Proc. 17th Inter. Joint Conf. on Artificial Intelligence (IJCAI'01)*, 2001.

[9] 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.

[10] C. A. Knoblock. Learning abstraction hierarchies for problem solving. In *Proc. 7th Nat. Conf. AI.(AAAI'90)*, 1990.

[11] 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.

[12] D. Long and M. Fox. Extracting route-planning:first steps in automatic problem decomposition. *AIPS'2000: Workshop on Analyzing and Exploiting Domain Knowledge*, 2000.

[13] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. In *Artificial Intelligence*, pages 115–135, 1974.