INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, Mi 48106-1346 USA
800-521-0600

UMI®
PLANIFICATION PAR DES ORDRES PARTIELS

par

Mohamed Mehdi Najjar

mémoire présenté au Département de mathématiques et d'informatique en vue de l'obtention
du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOK

Sherbrooke, Québec, Canada, décembre 2000
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
SOMMAIRE

Le nombre considérable d’essais traitant du problème de planification en intelligence artificielle a conduit à la naissance de plusieurs techniques comme la planification par l’exploration de l’espace d’états.


Certaines idées ont prouvé que l’exploration de l’espace d’états pourrait être contrôlée en évitant d’explorer tous les entrelacements des événements concurrents. Le principe est de considérer ces entrelacements comme un seul ordre partiel dans lequel l’ordre d’exécution des événements concurrents importe peu. Parmi tous les entrelacements possibles, il est suffisant d’en explorer un seul. Étant donné que tous les autres entrelacements représentent des occurrences d’un même ordre partiel, le contenu informationnel global reste invariant quel que soit l’ordre d’exécution des événements concurrents. C’est le principe des ordres partiels.

Le principe des ordres partiels s’est montré fort efficace pour la réduction de l’espace d’états dans la vérification des modèles de systèmes concurrents. Nous nous proposons de l’évaluer en planification en le greffant dans une approche de planification existante et populaire en intelligence artificielle (Graphplan).
Le 4 avril 2001, le jury suivant a accepté ce mémoire dans sa version finale.

Président-rapporteur: M. André Mayers
Département de mathématiques et d'informatique

Membre: M. Roger Nkambou
Département de mathématiques et d'informatique

Membre: M. Froduald Kabanza
Département de mathématiques et d'informatique
REMERCIEMENTS

Je tiens à exprimer ma sincère gratitude ainsi que mon entière reconnaissance envers mon directeur de recherche, le professeur Froduald Kabanza, pour son aide capitale et ses conseils, qui m'ont été fort utiles à la réalisation de ce travail.

Je tiens également à remercier Sonia Larochelle et le professeur André Mayers pour leurs suggestions concernant le style de la rédaction et de la présentation de ce mémoire.

Je voudrais enfin profiter de cette occasion pour dédier mon travail à mes parents avec tout mon amour.
TABLE DES MATIÈRES

SOMMAIRE ................................................................. ii

REMERCIEMENTS ................................................... iii

TABLE DES MATIÈRES ........................................ iv

LISTE DES TABLEAUX ........................................ vii

LISTE DES FIGURES ............................................... ix

INTRODUCTION ..................................................... 1

CHAPITRE 1 – GRAPHPLAN .......................................... 10

1.1 Le graphe de synthèse ........................................ 10

1.2 Les relations d'exclusions mutuelles ..................... 12

1.3 La propagation des relations d'exclusions mutuelles ........ 15

1.4 Description de l'algorithme ................................ 15

1.5 Exemple .......................................................... 21

1.6 L'ordonnancement des objectifs ............................. 23

1.7 Résultats expérimentaux ..................................... 24

1.8 Test d'arrêt ....................................................... 26

1.9 Les limites de Graphplan ..................................... 26
CHAPITRE 2 – LES ORDRES PARTIELS .............................................................. 29

2.1 La recherche classique ................................................................. 29
2.2 Les actions indépendantes ......................................................... 31
2.3 Les ordres partiels ................................................................. 31
2.4 La recherche sélective ............................................................. 33
2.5 Les ensembles persistants ....................................................... 34
2.6 Les ensembles ‘sleep’ .............................................................. 38
2.7 Combiner les ensembles ‘sleep’ avec les ensembles persistants ...... 44

CHAPITRE 3 – UNE MÉTHODE HYBRIDE ..................................................... 46

3.1 La création du graphe ............................................................... 47
3.2 La planification faite par Graphplan ........................................ 50
3.3 Le nouveau processus de planification ..................................... 51
3.4 Résumé ............................................................................. 55

CHAPITRE 4 – ÉVALUATION EXPÉRIMENTALE ........................................ 56

4.1 Le ‘rocket domain’ ................................................................. 57
4.2 Le ‘logistic domain’ .............................................................. 64
LISTE DES TABLEAUX

1. ‘rocket domain’ – exemple de buts reliés à la complexité .................. 59-60
2. ‘rocket domain’ – les temps d’exécution pour une complexité C1 .......... 60
3. ‘rocket domain’ – les temps d’exécution pour une complexité C2 .......... 61
4. ‘rocket domain’ – les temps d’exécution pour une complexité C3 .......... 61
5. ‘rocket domain’ – les temps d’exécution pour une complexité C4 .......... 61
6. ‘rocket domain’ – les temps d’exécution pour une complexité C5 .......... 62
7. ‘rocket domain’ – les temps d’exécution pour une complexité C6 .......... 62
8. ‘rocket domain’ – les temps d’exécution pour une complexité C7 .......... 62
9. ‘rocket domain’ – comparaison des temps d’exécution par rapport à la complexité ......................................................... 63
10. ‘logistic domain’ – exemple de buts reliés à la complexité .................. 66-68
11. ‘logistic domain’ – les temps d’exécution pour une complexité C1 .......... 68
12. ‘logistic domain’ – les temps d’exécution pour une complexité C2 .......... 69
13. ‘logistic domain’ – les temps d’exécution pour une complexité C3 .......... 69
14. ‘logistic domain’ – les temps d’exécution pour une complexité C4 .......... 69
15. ‘logistic domain’ – les temps d’exécution pour une complexité C5 .......... 70
16. ‘logistic domain’ – les temps d’exécution pour une complexité C6 .......... 70
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Représentation des opérateurs du 'rocket domain' selon le formalisme UCPOP</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Un exemple de graphe de synthèse pour un cas dans le 'rocket domain'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>Algorithme 1–La création du graphe</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>Algorithme 2–La recherche de plan</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>Le graphe de synthèse pour un exemple dans le 'rocket domain'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>Le temps d'exécution de Graphplan pour le 'rocket domain'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>Algorithme 3–La recherche classique</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>Entrelacement d'actions indépendantes dans un ordre partiel</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>9.</td>
<td>Algorithme 4–Une recherche sélective avec les ensembles persistants</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>10.</td>
<td>Méthode de calcul de l'ensemble persistant</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>11.</td>
<td>Algorithme 5–La procédure 'Persistent_Set'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>12.</td>
<td>Algorithme 6–Le calcul des ensembles 'sleep' selon la technique [GK91]...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>13.</td>
<td>Premier cas de calcul des ensembles 'sleep'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>14.</td>
<td>Deuxième cas de calcul des ensembles 'sleep'</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>15.</td>
<td>Utilisation des ensembles 'sleep' pour réduire le nombre d'actions à exécuter</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>16.</td>
<td>Algorithme 7 – Une recherche sélective avec une combinaison entre les ensembles persistants et les ensembles ‘sleep’</td>
</tr>
<tr>
<td>17.</td>
<td>Algorithme 8–La création du graphe</td>
</tr>
<tr>
<td>18.</td>
<td>Algorithme 9–La recherche de plans par Graphplan</td>
</tr>
<tr>
<td>19.</td>
<td>Algorithme 10 –La nouvelle procédure principale de recherche</td>
</tr>
<tr>
<td>20.</td>
<td>Algorithme 11–La création des successeurs</td>
</tr>
<tr>
<td>21.</td>
<td>Algorithme 12 – Le processus global de la planification</td>
</tr>
<tr>
<td>22.</td>
<td>‘rocket domain’ – spécification des opérateurs</td>
</tr>
<tr>
<td>23.</td>
<td>‘rocket domain’ – un exemple de conditions initiales aléatoires</td>
</tr>
<tr>
<td>24.</td>
<td>‘rocket domain’ – courbes des tems d’exécution par rapport à la complexité</td>
</tr>
<tr>
<td>25.</td>
<td>‘logistic domain’ – spécification des opérateurs</td>
</tr>
<tr>
<td>26.</td>
<td>‘logistic domain’ – un exemple de conditions initiales aléatoires</td>
</tr>
<tr>
<td>27.</td>
<td>‘logistic domain’ – courbes des tems d’exécution par rapport à la complexité</td>
</tr>
<tr>
<td>28.</td>
<td>‘repairer domain’ – spécification des opérateurs</td>
</tr>
<tr>
<td>29.</td>
<td>‘repairer domain’ – un exemple de conditions initiales aléatoires</td>
</tr>
<tr>
<td>30.</td>
<td>‘repairer domain’ – courbes des tems d’exécution par rapport à la complexité</td>
</tr>
<tr>
<td>31.</td>
<td>Un exemple d’exécution du planificateur sur un cas de problème dans le ‘rocket domain’</td>
</tr>
</tbody>
</table>
INTRODUCTION

Ce mémoire de maîtrise en informatique a pour objectif d'implémenter et d'évaluer une nouvelle idée de planificateur. Cette idée consiste à combiner une approche de planification existante en intelligence artificielle (Graphplan) avec une technique d'exploration de transitions partiellement ordonnées (Ordres partiels) habituellement utilisée pour la vérification des modèles de systèmes concurrents. Le but est de vérifier si cette combinaison donnerait un planificateur plus efficace que le planificateur Graphplan. Par 'plus efficace', nous voulons dire que le nouveau planificateur calculera des plans d'actions plus rapidement. Donc c'est une efficacité en temps d'exécution.

1. Problème de planification en intelligence artificielle

Dans notre vie quotidienne, nous planifions souvent des actions que nous allons faire pour atteindre un objectif spécifique. Pour un voyage en Floride, par exemple, nous pouvons planifier le retrait d'une certaine somme d'argent, la réservation des billets du voyage, l'heure d'arrivée à la gare, ensuite à l'aéroport, et ainsi de suite. Bien sûr, si nous planifions, c'est que c'est nécessaire. Dans les systèmes informatiques aussi bien que chez les humains, toutes les tâches ne peuvent pas être effectuées par des instructions codées en avance. Une partie des tâches, en effet, requiert la capacité de planifier, c'est-à-dire, de générer en temps réel des plans d'actions pour de nouveaux objectifs ou pour des situations nouvelles pour lesquelles les plans actuels ne sont pas adéquats.
En robotique, par exemple, nous planifions les chemins que va suivre un robot ou les manipulations d'objets qu'il va faire. Les actions de base pourraient correspondre à des processus effectuant des tâches simples comme se déplacer vers un point donné, ramasser un objet, le relâcher ou prononcer une phrase via le haut-parleur. Un plan, dans ce cas, serait une combinaison de ce genre d'actions pour accomplir une tâche donnée. Dans le domaine des protocoles de communication, les actions de base pourraient correspondre aux primitives de communication et de synchronisation. Un plan serait alors *un protocole*, c'est à dire un ensemble de règles pour fournir un service donné.

Par ailleurs, la planification est potentiellement très utile dans d'autres domaines, comme le transport postal où nous pouvons planifier les routes que suivent les ramasseurs de colis dans une compagnie comme *Federal Express* ou *UPS*. Il est vrai que c'est un problème traditionnel de recherche opérationnelle avant tout, mais la planification peut être aussi utilisée comme méthode de solution. Dans le multimédia également, nous pouvons planifier des présentations multimédias faites par des systèmes d'apprentissage (les systèmes tutoriels intelligents).

Un plan est un programme. Le mot *plan* suggère simplement que ce programme est dédié à des situations prévues, le plus souvent à base des prédictions de ce qui va ou pourrait se passer dans le système ou dans son environnement, d'où le terme *planifier*. C'est-à-dire, calculer des plans d'actions à exécuter à l'avenir dans des situations qu'on prévoit.

Dans sa formulation classique, le problème de planification en intelligence artificielle consiste à trouver une séquence finie d'actions qui transforment un état initial en un état qui satisfait un ensemble d'objectifs préfixés. Généralement, un problème de planification est constitué par (1) un ensemble d'opérateurs, (2) un ensemble d'objets, (3) un ensemble de propositions de départ et (4) un ensemble d'objectifs. Les objectifs constituent les propositions finales qui devraient être vraies à la fin du plan synthétisé. Les propositions de départ forment les conditions initiales du problème. Une instance atomique indivisible d'un opérateur appliquée à un ou à plusieurs objets est définie comme une action. Par exemple, une instantiation de l'opérateur *put (x) into (y)* donnerait l'action *put Objet1 into Contenaire2*. Dans un contexte usuel, l'état initial et l'état 'but' sont spécifiés par une liste de propositions. Les
actions sont représentées au moyen d'une représentation STRIPS [FN71]. Une action effectuée à un temps donné ajoute à l'environnement de nouvelles propositions qui sont les effets de l'action et supprime d'anciennes propositions en contradiction avec les nouveaux effets ajoutés formant la post-condition de l'action. De même, la pré-condition d'une action est un ensemble de propositions à partir desquelles une action peut être accomplie. C'est la condition permettant l'exécution de l'action.

Le nombre considérable d'essais traitant du problème de planification a conduit à la naissance de plusieurs techniques. Nous nous intéressons – dans le cadre de notre travail – à deux approches qui sont : (1) la planification par l'exploration de l'espace d'états et (2) la planification par les graphes de synthèse.

Dans la planification par l'exploration de l'espace d'états, tous les états possibles et imaginables constituent l'espace d'états formalisé par un graphe où un nœud correspond à un état et les branchements correspondent aux entrelacements possibles entre les différentes actions pouvant être exécutées à partir de cet état. Une recherche classique de plan serait d'explorer de façon récursive tous les états successeurs de tous les états rencontrés durant la recherche en débutant à partir de l'état initial et en exécutant toutes les actions activées à chaque état.

Dans la deuxième approche, un graphe de synthèse paraît similaire à un graphe d'espace d'états, sauf qu'il est basé non seulement sur l'aspect informationnel du système, mais qu'il tient compte aussi des objectifs à atteindre et des conditions initiales en plus de la notion du temps. Les graphes de synthèse traduisent le problème de la planification de telle sorte que les contraintes usuelles liées au problème deviennent explicitement résolues après un certain temps.

Graphplan est un planificateur proposé par A. Blum et M. Furst [BF97], qui génère un graphe de synthèse à partir des conditions initiales en appliquant des actions de base dont il dispose. Pour trouver un plan d'actions – s'il en existe un – menant aux objectifs prédéfinis, le planificateur applique un ensemble de règles.
2. Graphplan

Ce planificateur comporte deux phases principales : (1) la construction du graphe et (2) la recherche de plan à travers ce graphe. Le graphe de synthèse construit par Graphplan est ordonné par niveau. Il contient des niveaux de propositions alternés avec des niveaux d’actions. Le premier niveau du graphe est un niveau de propositions représentant les conditions initiales. La recherche de plan se fait avec un chainage arrière, depuis les objectifs formulés sur le graphe, vers les conditions initiales qui figurent dans le premier niveau de propositions. Le plan serait un ensemble ordonné d’actions (séquence) qui rend les objectifs préalablement fixés vrais à une étape finale. Les actions d’un plan sont accomplies – chacune ou certaines d’entre elles – à une étape bien définie. Dans un plan, une action peut être exécutée à une étape si ce même plan rend toutes ses pré-conditions vraies à cette étape.

Graphplan a été évalué – selon certaines études – comme l’un des meilleurs outils contribuant à l’évolution et à l’amélioration du paradigme de la planification [KEE97]. Le planificateur mérite son succès grâce à sa rapidité d’exécution.

3. La planification par l’exploration à chainage avant

Théoriquement, si l’espace d’états global est fini, il est possible de l’explorer en entier afin de trouver un état satisfaisant les objectifs. Mais en pratique, ce n’est pas toujours le cas, car la taille d’un espace d’états est trop grande pour qu’il puisse être exploré d’une façon exhaustive. Ce phénomène est appelé l’explosion d’états ou encore l’explosion combinatoire. Une des causes de l’explosion combinatoire est l’abondance des entrelacements entre les actions concurrentes. En effet, tous les entrelacements entre toutes les actions concurrentes figurent dans l’espace d’états global. Plusieurs idées ont été proposées pour remédier au problème d’explosion combinatoire. Tout en étant différentes, leur objectif est unique : éviter l’exploration anarchique de l’espace d’états, donc contrôler la recherche.
F. Bacchus et F. Kabanza [BK2000] ont proposé de contrôler l’exploration de l’espace d’états avec un chaînage avant au moyen d’un mécanisme de contrôle de la recherche basé sur la connaissance du domaine, qui est représentée par des formules de logique temporelle du premier ordre. Cette méthode a été implémentée dans TLPLAN. L’évaluation expérimentale de ce planificateur a montré qu’en dépit de la simplicité de la conception d’une recherche en profondeur avec un chaînage avant, cette technique classique peut devenir très puissante si elle est combinée avec d’autres stratégies efficaces de contrôle de recherche.

D’autre part, S. Kambhampati et al. [KEE97] ont utilisé l’exploration à chaînage avant pour rechercher un plan d’actions à partir d’un graphe de synthèse généré par Graphplan. L’idée consiste à structurer le graphe de synthèse comme s’il était un espace d’états compact et de rechercher un plan avec un chaînage avant en profondeur. En considérant un état comme un ensemble de propositions, chaque niveau propositionnel est vu comme un ensemble d’états et chaque niveau d’actions contient toutes les actions qui peuvent être activées à partir des propositions de l’ensemble des états. Le graphe de synthèse est vu ainsi comme une représentation compacte d’un espace d’états global. Les tests expérimentaux de cette approche ont montré que dans certains domaines d’étude, la recherche avec un chaînage avant s’avère plus efficace que Graphplan, surtout quand le nombre d’actions pouvant être exécutées à un niveau donné est très important mais que seules quelques-unes sont vraiment appropriées pour être retenues dans un plan.

Par ailleurs, P. Godefroid [God94,God96,God97] a montré que l’exploration de l’espace d’états pourrait être contrôlée en évitant d’explorer tous les entrelacements des actions concurrentes. L’idée est de considérer ces entrelacements comme un seul ordre partiel dans lequel l’ordre d’exécution des actions concurrentes importe peu. Parmi tous les entrelacements possibles entre les actions concurrentes, la technique vise à en explorer un seul. Étant donné que tous les autres entrelacements représentent des occurrences d’un même ordre partiel, le contenu informationnel global reste invariant quel que soit l’ordre d’exécution des actions concurrentes.
Cette approche a été implémentée dans des systèmes de vérification de protocoles (SPIN, VeriSoft). Pour la plupart des exemples étudiés, la méthode des ordres partiels a révélé une nette efficacité dans la réduction de l’espace d’états. Notons enfin que cette approche des ordres partiels a été testée dans des vérificateurs de programmes et non pas dans des planificateurs.

Sur la même longueur d’onde, P. Godefroid et F. Kabanza [GK91] ont présenté également une simple recherche à chaînage avant qui se base sur l’exploration d’un seul entrelacement parmi ceux représentant un ordre partiel. L’idée principale est de construire un automate de traces qui génère un seul entrelacement d’actions concurrentes. Les essais réalisés sur des exemples de vérification ont montré que cette technique réduit considérablement l’espace d’états.

4. Objectif du mémoire

Comme nous l’avons présenté ci-dessus, l’objectif de ce mémoire est d’évaluer une nouvelle idée de planification basée sur une approche ‘hybride’. Cette approche est le résultat d’un amalgamé entre Graphplan et la théorie des ordres partiels appliquée dans la vérification des systèmes concurrents.

Il est juste de noter que notre espace d'états formulé à partir du graphe de synthèse n'est pas 'compact', c'est-à-dire que les états ne sont pas fusionnés en un ensemble d'états comme dans la méthode de S. Kambhampati et al.

5. Méthodologie

Pour la transformation du graphe de synthèse en un espace d'états, nous allons d'abord dresser un graphe de synthèse à l'aide du planificateur Graphplan. À partir des niveaux propositionnels du graphe, et en commençant par le premier contenant les conditions initiales, nous formulons les états. Pour chaque état donné, obtenu à partir d'un niveau de proposition (i), nous cherchons, dans le niveau d'actions (i+1) du graphe, les actions qui peuvent être activées depuis l'état donné. Par exemple, une action (a) est jugée comme activée à partir d'un état (s) si la pré-condition de l'action (a) est incluse dans l'ensemble des propositions de l'état (s).

Du moment où nous sommes capable de formuler des états à partir du graphe et d'obtenir les actions qui peuvent être activées à partir de chaque état, l'exploration de l'espace d'états s'effectue de la façon suivante : pour un état donné, parmi toutes les actions qui peuvent être activées à partir de cet état, nous construisons un ensemble contenant uniquement les actions retenues pour être exécutées et donner d'autres états. Toutes les autres actions applicables et n'appartenant pas à l'ensemble ainsi obtenu sont rejetées. L'ensemble est sélectionné selon une combinaison de deux techniques se basant sur les ordres partiels : (1) la technique des ensembles persistants et (2) celle des ensembles 'sleep'.

La génération du graphe à la première étape nous évite l'instanciation des opérateurs à chaque fois que nous obtenons l'ensemble des actions applicables à partir d'un état donné. Il suffit d'aller chercher ces actions dans le graphe, plus précisément dans le niveau d'actions adjacent au niveau de propositions dans lequel figure l'état.
6. Conséquences et résultats

Nous avons implémenté notre approche hybride en langage C et nous l'avons testée dans certains domaines d'étude. Les essais comparatifs entre Graphplan et notre planificateur ont montré que Graphplan demeure plus efficace pour le temps d'exécution.

Il est vrai que notre nouveau planificateur hybride est nettement meilleur qu'une simple recherche en profondeur à chaînage avant, car nous sommes parvenu à régler le problème de l'explosion combinatoire grâce aux techniques des ordres partiels. Donc, nous avons réussi à réduire considérablement – par rapport à la recherche classique – le temps d'exécution ainsi que l'espace mémoire utilisé pour le stockage des données.

Cependant, étant donné que notre planificateur n'utilise aucune heuristique et n'adopte aucune stratégie évoluée pour contrôler la recherche (comme l'utilisation des formules logiques d'optimisation pour TLPLAN), la recherche de plans – bien qu'elle se fasse dans un espace d'états réduit – reste encore quelque peu anarchique. Ceci s'explique par le fait que les actions retenues pour être appliquées (suite à la sélection faite au moyen des ordres partiels) sont exécutées selon un ordre aléatoire. Nous ne disposons pas d'un mécanisme de raisonnement capable de choisir une action particulière et de la favoriser – selon certains critères – par rapport aux autres actions retenues.

En dépit du résultat peu favorable de notre travail, nous avons pu constater le rôle capital des mécanismes évolués de contrôle de la recherche qui se basent sur le raisonnement et sur la connaissance acquise du domaine.

7. Organisation du mémoire

Le premier chapitre est consacré à Graphplan. Nous allons décrire d'une façon détaillée le fonctionnement du planificateur. Nous présentons ensuite une évaluation de Graphplan au moyen d'exemples dans un certain domaine d'étude (le 'rocket domain') et nous finirons par une description de ses limites.
Dans le Chapitre 2, nous abordons les ordres partiels et nous introduisons certains algorithmes relatifs aux différentes techniques de réduction de l'espace d'états que nous avons utilisées pour la planification avec les ordres partiels.

Le Chapitre 3 est dédié à notre technique hybride. Nous montrons en détail comment nous avons procédé pour combiner Graphplan avec les ordres partiels. Nous décrivons les modifications faites sur Graphplan en indiquant comment nous avons fait pour les réaliser.

L'évaluation approfondie de cette nouvelle approche est faite dans le Chapitre 4 où nous avons étudié trois domaines différents de manière empirique. Enfin, nous présentons la conclusion résumant tout notre travail ainsi que les possibles améliorations que nous pourrons faire dans l'avenir.
Chapitre 1

GRAPHPLAN

Le planificateur Graphplan proposé par A. Blum et M. Furst [BF97] génère un graphe de synthèse à partir des conditions initiales en appliquant les actions de base dont il dispose. À partir du graphe, il essaye – en appliquant certaines règles – de trouver une séquence d’actions menant aux objectifs prédéfinis.

1.1 Le graphe de synthèse

Le graphe de synthèse construit par Graphplan est un graphe par niveau constitué de deux types de nœuds et de trois types d’arcs. Les niveaux du graphe sont de deux sortes : des niveaux de propositions alternés avec des niveaux d’actions. Les niveaux de propositions contiennent les nœuds de propositions, alors que les niveaux d’actions sont formés par les nœuds d’actions.

Le premier niveau du graphe est un niveau de propositions où chaque nœud y figurant représente une proposition existante parmi les conditions initiales. Un graphe de synthèse est résumé comme suit : des propositions vraies à un temps (t), des actions possiblement applicables à un temps (t), des propositions possiblement vraies à un temps (t+1), des actions possiblement applicables à un temps (t+1), des propositions possiblement vraies à un temps (t+2), et ainsi de suite.
Les arcs dans le graphe schématisent les relations entre les actions et les propositions. Un nœud d'action dans un niveau d'action (i) est connecté par : (1) des arcs de pré-conditions à ses nœuds de pré-conditions situées dans le niveau de propositions (i), (2) des arcs d'effets ajoutés à ses nœuds de post-conditions qui sont dans le niveau de propositions (i+1), (3) des arcs d'effets supprimés à ses effets supprimés dans le niveau de propositions (i+1).

Dans le graphe, une action peut exister dans un niveau d'action (i) à condition que toutes les propositions de sa pré-condition existent dans le niveau de propositions (i). Un cas particulier d'actions dites 'no-op' existe dans tous les niveaux du graphe. Une action 'no-op' est une action neutre. Sa post-condition est exactement identique à sa pré-condition.

Une proposition (p) ne peut exister dans un niveau de propositions (i+1) que si et seulement si elle fait partie de la post-condition d'une action figurant dans le niveau d'actions (i). Remarquons que cette même proposition (p) pourrait aussi représenter en même temps un effet supprimé d'une autre action de niveau (i).
1.1.1 Exemple

Supposons que nous avons des cargos de marchandise à déplacer de ville en ville et que nous disposons d'un certain nombre d'avions pour le faire. Un cargo peut être chargé (load) dans un avion - respectivement, déchargé (unload) de l'avion - si l'avion et le cargo se trouvent à un même emplacement. L'avion initialement rempli de kérosène (has fuel) épuise son combustible en se déplaçant (move). Le 'rocket domain' introduit par Veloso en 1989 [Vel89] est formé de ces trois opérateurs : 'load', 'unload' et 'move'. Un problème typique dans ce domaine se défini comme suit : étant donné un ou plusieurs avions et un ou plusieurs cargos à un emplacement de départ, l'objectif est de faire déplacer les cargos à un certain nombre de destinations. Les opérateurs de ce domaine sont représentés dans la figure 1 selon le formalisme UCPOP présenté par A. Barrett et D. Weld [BW94].

Supposons que nous avons un avion 'R', deux cargos 'A' et 'B', un emplacement de départ 'L' et une destination 'P'. Le but étant de déplacer 'A' et 'B' vers 'P'. Le graphe de synthèse est représenté par la figure 2 où les arcs des effets supprimés sont en pointillés alors que ceux des effets ajoutés sont en traits continus et les 'no-op' sont les nœuds noirs.

1.2 Les relations d'exclusions mutuelles entre nœuds

En formulant le graphe de synthèse, le travail le plus rigoureux à faire est la détection et la propagation des relations d'exclusions mutuelles entre les nœuds d'actions et de propositions.

1.2.1 Les actions exclusives

Deux actions sont 'mutuellement exclusives' dans un même niveau d'actions, si aucun plan
Figure 2 : Un exemple de graphe de synthèse pour un cas dans le 'rocket domain'.

valide ne peut les contenir simultanément dans ce même niveau. Par exemple, 'load(A,L,R)' et 'move(R,L,P)' sont mutuellement exclusives dans le niveau 2 (Figure 2). Plus précisément, deux actions 'a' et 'b' sont marquées exclusives dans un niveau d'actions (i), si on a une des deux conditions suivantes :

- la rivalité : il existe - au moins - une pré-condition de 'a' et une pré-condition de 'b' qui sont mutuellement exclusives dans un niveau de propositions donné.

- l'interférence (non indépendance) : chacune d'entre elles supprime la pré-condition ou la post-condition de l'autre.

Supposons par exemple – dans le 'rocket domain' – qu'un cargo 'Jason' se trouve initialement à Paris, qu'un autre cargo 'Alex' se trouve initialement à Londres, et que nous disposons d'un avion 'Rocket 1'. Les deux actions 'load Alex into Rocket 1 in London' et 'load Jason into Rocket 1 in Paris' sont exclusives à un même niveau, car la première a besoin de 'Rocket 1 at
Les propositions exclusives

Deux propositions sont "mutuellement exclusives" dans un même niveau, si aucun plan valide ne peut les rendre vraies toutes les deux dans ce même niveau. Par exemple, 'at R.P' et 'at R.L' sont mutuellement exclusives dans le niveau 2 (Figure 2). Généralement, deux propositions 'p' et 'q' sont marquées comme exclusives à un niveau donné, si toutes les actions pouvant générer 'p' sont exclusives avec toutes les actions qui peuvent générer 'q'. Les deux propositions 'Rocket1 at London' et 'Rocket1 at Paris' - par exemple - sont exclusives à un même niveau, car toutes les actions pouvant générer la première sont exclusives avec toutes les actions qui génèrent la deuxième. Plus précisément, deux propositions 'p' et 'q' sont marquées comme exclusives si toute action (a) ayant 'p' dans sa post-condition ( (a) et 'p' sont reliées par un arc d'effet ajouté) est exclusive avec toute action (b) ayant 'q' dans sa post-condition.

Deux propositions peuvent être mutuellement exclusives : (1) à tous les niveaux du graphe, ou (2) à partir d'un certain niveau, ou (3) jusqu'à un certain niveau. Par exemple, si les propositions 'Alex at London' et 'Rocket1 at London' figuraient parmi les conditions initiales, alors les propositions 'Alex in Rocket1' et 'Rocket1 at Paris' seraient exclusives au niveau 2 mais pas au niveau 3 car 'Rocket1 at Paris' pourrait être obtenue en exécutant 'move Rocket1 from London to Paris' au deuxième niveau d'actions. Considérons un autre exemple avec les
deux propositions initiales suivantes : 'Rocket1 at London' et 'has-fuel Rocket1'. Les deux actions 'move Rocket1 from London to Paris' et 'load Alex into Rocket1 in London' sont exclusives à un même niveau, car la première action supprime la proposition 'Rocket1 at London' faisant partie de la pré-condition de la deuxième action.

Graphplan utilise toutes ces règles – relatives aux actions et propositions exclusives – pour détecter et enregistrer les relations d'exclusions mutuelles entre les nœuds. La mémorisation sert à propager ces relations à travers les niveaux du graphe.

1.3 La propagation des relation d'exclusion mutuelles

Supposons qu'à un temps (t-1) un objet 'O' ne peut être qu'à un seul emplacement. Dans ce cas : (1) deux actions 'move' applicables à un temps (t-1) et faites à partir de deux emplacements différents seront mutuellement exclusives à cause de la concurrence pour besoin. Par exemple 'move Rocket1 from London to Paris' et 'move Rocket1 from Paris to New York'. (2) deux actions 'move' applicables à un temps (t-1) et faites à partir d'un même emplacement seront aussi mutuellement exclusives, car chacune d'entre elles supprime la pré-condition de l'autre (Par exemple 'move Rocket1 from London to Paris' et 'move Rocket1 from London to New York'). Donc, de ce fait, l'objet 'O' ne peut être qu'à un seul emplacement à un temps (t). En résumé, si un objet 'O' ne peut pas être simultanément à deux endroits à un temps (t-1), alors il ne peut pas être simultanément à deux endroits à un temps (t). En propageant ces contraintes à travers les niveaux au moment de la génération du graphe, le planificateur les prendra en compte au cours du processus de planification.

1.4 Description de l'algorithme

À un haut degré d'abstraction, l'idée de base est la suivante : en commençant avec un graphe de synthèse contenant initialement un seul niveau de propositions – qui résume les conditions initiales – Graphplan génère le niveau d'ordre (i) à partir de celui d'ordre (i-1). Il étend le
graphe jusqu'à ce que tous les buts préfixés figurent dans un niveau de propositions et qu'il n'y ait pas deux d'entre eux qui sont mutuellement exclusifs. C'est à ce moment que le processus de recherche d'un plan commence depuis les objectifs obtenus, et en allant vers l'arrière jusqu'aux conditions initiales.

1.4.1 La génération du graphe de synthèse

Généralement, pour un niveau de propositions (i), le planificateur vérifie si les objectifs sont atteints, au quel cas, il arrête la construction du graphe. Sinon, il génère le ième niveau d’actions.

1.4.1.1 La création d’un niveau d’actions

La création d’un niveau d’actions (i) – à partir de son niveau de proposition (i) – se fait de la façon suivante: (1) pour chaque opérateur ‘O’, il génère une instance de ‘O’ si toutes les propositions de sa pré-condition sont de niveau (i). Par exemple, à partir de la proposition ‘Rocket1 in London’ nous pouvons obtenir deux instances de l’opérateur ‘move’ qui sont : ‘move Rocket1 from London to Paris’ et ‘move Rocket1 from London to New York’. (2) Il insère toutes les actions ‘no-op’ et tous les arcs de pré-condition. (3) Il détermine les relations d’exclusion mutuelle (si elles existent) entre les actions générées en appliquant les règles d’interférence et de rivalité. Plus précisément, il établit pour chaque action, la liste des actions dont elle est exclusive. Ceci nous permettra de propager les relations entre les actions d’un niveau à un autre. (4) Il crée le (i+1)ième niveau de propositions et détermine les relations d’exclusions mutuelles entre ses propositions.

1.4.1.2 La création d’un niveau de proposition
Le niveau de propositions (i+1) est créé de la façon suivante : (1) Graphplan place toutes les post-conditions des actions formulées – y compris les 'no-op' – lors de la création du ième niveau d'actions ; il connecte ensuite les post-conditions à leurs actions avec les arcs appropriés. (2) Il examine les propositions obtenues deux à deux. Il marque deux propositions comme 'mutuellement exclusives' si toutes les actions dont la première proposition constitue une de leurs post-conditions, sont exclusives avec toutes les actions dont la deuxième proposition constitue une de leurs post-conditions.

1.4.1.3 Algorithme

L'algorithme général (Algorithme 1) de la création du graphe est décrit dans la figure 3. Comme nous l'avons mentionné précédemment, à partir d'un ensemble ($P_i$) de conditions initiales, d'une liste des opérateurs permis et d'un ensemble 'G' de buts prédéfinis, la création du graphe se fait par couches. Une couche (i) est composée d'un niveau (i) d'actions et d'un niveau (i+1) de propositions.

Toutes les conditions initiales sont mises au début dans une table de propositions 'Fact_Table' (ligne 3). Ensuite, tant que tous les buts ne sont pas obtenus de façon non exclusive (ligne 4), le planificateur continue à créer les couches avec la procédure 'Create_Graph_Layer' (ligne 5). D'abord, il commence par copier toutes les propositions du dernier niveau propositionnel (i) dans le niveau actuel de propositions (i+1) qu'il s'apprête à construire (ligne 6). Par la suite, il génère toutes les actions possibles à partir du niveau de propositions (i) ; elles formeront le niveau d'actions (i) (ligne 7). Les arcs de pré-condition sont placés entre les actions obtenues et leurs propositions de pré-conditions dans le niveau de propositions (i) (ligne 8). Puis, les post-conditions des actions seront mises dans le niveau (i+1) et les arcs de post-conditions ainsi que ceux des effets ajoutés seront également placés (lignes 9-11). Enfin, c'est le tour des actions 'no-op' (ligne 12). Ceci étant fait, les relations d'exclusions mutuelles entre les actions du niveau (i) et celle entre les propositions de niveau (i+1) seront formulées (lignes 13-14).
1. \textbf{Create}\_Graph (Operators, Pi, G) \{ 
2. \hspace{1em} time = 0 ; 
3. \hspace{1em} load (Pi) in Fact\_Table[time] ; 
4. \hspace{1em} while (G NOT reached) \{ 
5. \hspace{1em} \hspace{1em} \textbf{Create}\_Graph\_Layer (Operators) \{ 
6. \hspace{1em} \hspace{1em} \hspace{1em} copy facts from Fact\_Table[time] to Fact\_Table[time +1] ; 
7. \hspace{1em} \hspace{1em} \hspace{1em} Do\_Operators (Fact\_Table[time +1], Operators) ; 
8. \hspace{1em} \hspace{1em} \hspace{1em} add pre-condition\_edges ; 
9. \hspace{1em} \hspace{1em} \hspace{1em} put all post-conditions in Fact\_Table[time+1] ; 
10. \hspace{1em} \hspace{1em} \hspace{1em} add pre-condition\_edges ; 
11. \hspace{1em} \hspace{1em} \hspace{1em} add delete\_edges ; 
12. \hspace{1em} \hspace{1em} \hspace{1em} make all no-op ; 
13. \hspace{1em} \hspace{1em} \hspace{1em} find all mutex actions ; 
14. \hspace{1em} \hspace{1em} \hspace{1em} find all mutex facts ; 
15. \hspace{1em} \hspace{1em} \} 
16. \hspace{1em} \hspace{1em} time = time +1 ; 
17. \hspace{1em} \hspace{1em} if ( all (g) in G are independent in actual level) then 
18. \hspace{1em} \hspace{1em} \hspace{1em} G is reached ; 
19. \hspace{1em} \hspace{1em} \} 
20. \hspace{1em} \}

Figure 3 : Algorithme 1 – la création du graphe.

1.4.2 La recherche de plan

Après avoir formulé le graphe de synthèse et ayant obtenu des buts totalement indépendants, Graphplan commence à rechercher un plan en utilisant une stratégie à chainage arrière. Il opère niveau par niveau, en débutant à partir des objectifs et, en prenant en compte les contraintes d’exclusions mutuelles – déjà établies au cours de la phase de la création du graphe – il essaie d’atteindre les conditions initiales.

1.4.2.1 Méthodologie de recherche

Après avoir construit le graphe de synthèse et après avoir obtenu des objectifs totalement indépendants au dernier niveau de propositions, le planificateur fait appel à la procédure
1. Do_Plan (G, time_step) {
2.   True_Goals = ∅;
3.   while (G ≠ ∅) do {
4.     for all (g) in G do {
5.       if ((g) ∈ True_Goals ) then
6.         G = G - g ;
7.       else {
8.         Bad_Ops = ∅; Sub_Goals = ∅; Good_Ops = ∅;
9.         for all (op) \ (g) ∈ post(op) do {
10.        Mark_exclusive (op) ;
11.        for all (op*) \ op* ∈ Good_Ops
12.         if (op is exclusive with op*)
13.         then Bad_Ops = Bad_Ops + op ;
14.        else {
15.         for all (g') \ (g') ∈ pré(op) do {
16.         if ( g' ∈ Sub_Goals ) then
17.         Sub_Goals = Sub_Goals + g' ;
18.         }
19.         for all (g*) \ (g*) ∈ post(op) do {
20.         if ( g* ∈ True_Goals ) then
21.         True_Goals = True_goals + g* ;
22.         }
23.         Good_Ops = Good_Ops + op ;
24.         }
25.       }
26.     }
27.     G = Sub_Goals ;
28.   }
29.   Sub_Goals = ∅;
30.   time_step = time_step - 1;
31.   if (time_step > 0) then
32.     Do_Plan (G, time_step) /* appel recursif */
33. }
34. }

Figure 4 : Algorithme 2 – la recherche de plan.

‘Do_Plan’ pour chercher un plan en débutant à partir des buts et en essayant d’atteindre les conditions initiales du premier niveau propositionnel.

L’Algorithme 2 de la figure 4 résume la procédure ‘Do_Plan’ à la quelle Graphplan fait appel pour rechercher un plan.
Etant donné un ensemble ‘G’ de buts à un niveau (t), Graphplan essaie de trouver un ensemble d’actions – ‘no-op’ incluses – de niveau (t-1) telle que leurs post-conditions soient dans ‘G’ (lignes 4 et ligne 9). Les pré-conditions des actions choisies forment l’ensemble ‘Sub_Goals’ des sous buts de niveau (t-1) (lignes 15-17). En acceptant une action donnée, toutes les propositions constituant sa post-condition seront considérées comme vraies et seront placées dans l’ensemble True_Goals’ (lignes 19-21). Pour chaque objectif – à examiner – de niveau (t), une action de niveau (t-1) l’accomplissant est sélectionnée à condition qu’elle ne soit pas exclusive avec toutes les autres actions retenues dans l’ensemble ‘Good_Ops’ contenant les actions menant aux objectifs déjà examinés (ligne 11). Une fois que tous les buts de niveau (t) ont été examinés et que l’ensemble ‘Good_Ops’ des actions indépendantes qui les accomplissent a été formulé, toutes les pré-conditions de ces actions (qui figurent dans l’ensemble ‘Sub_Goals’) sont prises comme de nouveaux objectifs à examiner au niveau (t-1) avec la même méthode récursive (lignes 29-33).

Graphpan réussit à trouver un plan s’il parvient à atteindre les conditions initiales (suite au même appel récursif de ‘Do_Plan’). Deux autres propriétés importantes ajoutées à cette stratégie, mettent en valeur l’originalité de Graphplan.

1.4.3 Originalités

1.4.3.1 La vérification des courts-circuits

Une vérification ‘en avant’ est employée à chaque fois qu’une action est retenue au cours de la phase de l’examen des buts à un niveau donné. Graphplan s’assure en acceptant une action (a) qu’aucun des autres buts restant parmi la liste des buts à examiner ne sera court-circuité, c’est à dire, il vérifie pour chaque but restant s’il existe au moins une action pouvant le créer qui n’est pas exclusive avec toutes les actions retenues (pour accomplir les buts déjà examinés) y compris l’action (a), dans le cas contraire, l’action (a) sera refusée car elle va
rendre au moins un des buts restant impossible à accomplir. Lors d’une telle situation, le planificateur choisira une autre action en faisant appel à la méthode récursive.

1.4.3.2 La mémorisation des échecs

À chaque fois que Graphplan conclut qu’un ensemble de sous buts ne peut pas être satisfait à un niveau (t) – à cause des exclusions mutuelles entre les sous buts ou entre les actions les accomplissant – il enregistre cet ensemble de sous buts et le niveau (t) dans une table. Quand le planificateur formule un ensemble de sous buts à examiner à un niveau donné et avant de faire appel à la méthode récursive pour sélectionner les actions, il consulte sa table pour voir s’il a déjà rencontré un tel ensemble à ce même niveau, auquel cas, il arrête sa recherche.

1.5 Exemple

Considérons encore une fois, un problème dans le ‘rocket domain’ avec les conditions initiales suivantes :

- deux avions : Rocket1 et Rocket2
- trois cargos : A, B et C
- un emplacement de départ S
- trois objectifs : A à un emplacement X, B au même emplacement X et C à un emplacement Y

Le graphe de synthèse – présenté dans la figure 5 – est composé de trois niveau d’actions. Pour rechercher un plan, d’abord, le planificateur sélectionnera un premier but, par exemple ‘A
at X' et choisira une action de niveau 2 qui le rend vrai, comme 'unload A from Rocket1 at X'.

Toutes les actions exclusives avec celle-ci seront marquées comme non acceptables au niveau 2 (comme 'unload C from Rocket1 at Y'). Le planificateur choisira par la suite un autre but, comme 'B at X'. S'il décide de sélectionner 'unload B from Rocket2 at X' au niveau 2 pour accomplir cet objectif, il remarquera par la vérification des courts-circuits que le but 'C at Y'
sera court-circuité, car toutes les actions qui le rendent vrai sont exclusives avec 'unload B from Rocket2 at X' et 'unload A from Rocket1 at X'. Donc, cette action ne sera pas choisie. Il sélectionnera par contre, une autre action comme 'unload B from Rocket1 at X' pour accomplir 'B at X' sans court-circuiter 'C at Y'. Une fois que le planificateur a terminé de choisir les actions de niveau 2, il examinera l'ensemble des sous buts de niveau 2 (qui sont les pré-conditions des actions sélectionnées) comme 'Rocket1 at X' et 'A in Rocket1'.

1.6 L'ordonnancement des objectifs

L'examen des buts au cours du processus de planification néglige leur ordonnancement. Les buts sont choisis de façon aléatoire. Une caractéristique qui s'ajoute à la méthodologie adoptée par Graphplan est la notion des ensembles minimaux d'actions choisies. Grâce à cette notion, l'ordre de sélection des objectifs importe peu.

1.6.1 Les ensembles minimaux

Soit 'G' un ensemble de buts formés à un niveau (t).

Définition 1.1 : Un ensemble A d'actions de niveau (t-1) accomplissant 'G' est dit minimal si :

- Chaque but de 'G' est une post-condition d'une action de A.
- Aucune action ne peut être soustraite de A sans que sa post-condition ne soit soustraite de 'G'.

III.
Soient, par exemple, deux buts ‘g1’ et ‘g2’ et deux actions (a1) et (a2) telle que (a1) accompli ‘g1’ et (a2) accompli ‘g1’ et ‘g2’, l’ensemble A formé de (a1) et (a2) n’est pas minimal, un ensemble minimal serait formé uniquement de (a2).

La méthode récursive d’examen des buts tend à former des ensembles minimaux d’actions. Etant donné que toutes les actions de l’ensemble minimal seront prises en compte pour formuler les sous buts, l’ordre des buts importe peu.

1.7 Résultats expérimentaux

À titre indicatif, la figure 6 montre le temps d’exécution de Graphplan pour le ‘rocket domain’. Les conditions initiales étaient d’avoir trois emplacements (Londres, Paris et New York), deux avions (Rocket1 et Rocket2) et N cargos. Tous les objets sont initialement à Londres et les avions sont pleins de kérosène, le but était de faire déplacer \([N/2]\) des cargos à Paris et \([N/2]\) à New York.

Notons que dans ce cas, le graphe de synthèse se compose uniquement de trois niveaux. Tous les buts – indépendamment du nombre N des cargos – sont obtenus de façon non exclusives au troisième niveau de propositions. Le temps d’exécution de Graphplan est quasiment linéaire par rapport au nombre de buts.

Dans la plupart des cas et quelque soit le domaine d’étude, Graphplan s’est montré efficace. Comme nous l’avons mentionné précédemment, quatre facteurs majeurs sont à l’origine de son efficacité : (1) les relations d’exclusion mutuelles, (2) la prise en compte des plans parallèles, (3) la mémorisation des échecs rencontrés et (4) l’usage du graphe de plans.

1.7.1 les relations d’exclusions mutuelles

Ces relations sont capables de représenter la majeure partie des contraintes liées aux
Figure 6 : Le temps d’exécution de Graphplan pour le ‘rocket domain’.

problèmes telle que : (1) la rivalité, (2) la concurrence pour besoin et (3) l’interférence (voir section 1.2).

1.7.2 La prise en compte des plans parallèles

Dans certains cas, comme le ‘rocket domain’ par exemple, le plan valide contient des sous plans parallèles (voir section 2 de l’introduction). Toutes les actions indépendantes appartenant chacune à un sous plan parallèle sont choisies – en même temps – dans le même
niveau d’actions. La recherche de plan est relativement rapide par rapport à d’autres stratégies qui reposent sur l’ordonnancement des buts.

1.7.3 Le mémorisation des échecs rencontrés

En gérant une table contenant la liste des objectifs non réalisables à un niveau donné, Graphplan évite beaucoup de travail redondant.

1.7.4 L’usage du graphe de plans

Graphplan réalise un gain énorme en temps en décidant de construire le graphe de synthèse à la première phase avant la recherche de plan. Une fois le graphe créé, il n’a plus besoin d’instancier les actions à partir des opérateurs au cours de la planification, car toutes les actions dont il pourrait avoir besoin ont été formulées au moment de la création du graphe.

1.8 Le test d’arrêt

Considérons un cas particulier de problèmes où il est impossible de trouver un plan (à cause de certains buts non réalisables en même temps comme ‘A at Paris’ et ‘A at New York’), à un moment donné – au cours de la création du graphe – il y aura un niveau de propositions ‘p’ à partir duquel tous les niveaux de propositions suivants lui seront identiques. Ils contiendront le même ensemble de propositions avec les mêmes relations d’exclusions mutuelles. À cause des ‘no-op’, si une proposition apparaît dans un niveau (i), alors elle apparaîtra dans tous les niveaux de propositions qui suivront ce niveau. Etant donné que les ensembles de propositions que nous pouvons créer avec des opérateurs définis en format STRIPS sont des ensembles finis, il existera un niveau de proposition ‘p’ à partir duquel tous les niveaux suivants lui seront exactement identiques. Plus généralement, si deux niveau de propositions adjacents
'p1' et 'p2' sont exactement identiques (avec les mêmes relations d'exclusions mutuelles) alors tous les niveaux de propositions qui suivront 'p2' seront identiques à 'p1' et 'p2'. Dans ce cas, nous disons qu'il y a une redondance de niveaux.

Si Graphplan détecte une redondance de niveaux pendant la création du graphe, alors il conclut que le problème ne sera pas résolu et il arrête l'exécution car il est impossible de trouver un plan d'action menant aux buts prédéfinis.

1.9 Les limites de Graphplan

En dépit de l'originalité et de l'efficacité de Graphplan, celui-ci montre quelques limites.

1.9.1 Une forte liaison au format STRIPS

Graphplan ne peut opérer que sur des cas de problème définis en format STRIPS. Dans ce type de formalisme, les actions ne peuvent pas créer de nouveaux objets. Les effets obtenus (post-conditions) par l'exécution des actions doivent être déterminés statiquement. Dans beaucoup de domaines naturels et artificiels, il existe des situations qui ne respectent pas cette contrainte.

Supposons par exemple, que l'on dispose de l'action 'peindre tout en rouge dans la chambre', et prenons le cas d'une chambre dans laquelle se trouve initialement : une table et deux chaises. Si nous appliquons l'action 'peindre tout en rouge dans la chambre' à partir de ces conditions initiales, nous aurons comme post-condition : une table peinte en rouge et deux chaises peintes en rouge. Si maintenant nous exécutons d'abord l'action 'faire sortir les chaises de la chambre', ensuite l'action 'peindre tout en rouge dans la chambre', nous aurons comme post-condition dans ce cas : une table peinte en rouge. Donc, l'ensemble des objets qui seront peints en rouge dépend de ce qu'il y a dans la chambre au moment de l'exécution de l'action 'peindre tout en rouge dans la chambre'. La définition des opérateurs dans le format STRIPS nécessite une spécification statique des paramètres (objets), de la pré-condition et de
la post-condition. Or, les effets de l’action ‘peindre tout en rouge dans la chambre’ – par exemple – ne peuvent pas être déterminés de façon statique. Nous pouvons formaliser sa post-condition en utilisant des effets conditionnels, mais le format STRIPS demeure incapable de définir une telle action.

1.9.2 Sans ses atouts

Le planificateur utilise son habilité à : (1) déterminer les relations d’exclusion mutuelles pour essayer de cerner les contraintes du problème sur lequel il opère et (2) détecter les sous plans parallèles composés d’actions indépendantes. Il est vrai que dans la majorité des cas – surtout dans les domaines naturels – ces deux atouts facilitent la tâche de Graphplan. Mais dans certains cas où il n’y a pas de plans parallèles et où les relations d’exclusions mutuelles s’avèrent peu efficaces pour la résolution, le planificateur, en absence d’un raisonnement substituant, fonctionne d’une façon ad – hoc, ce qui lui fait perdre beaucoup de son efficacité.
Chapitre 2
LES ORDRES PARTIELS

Comme nous l'avons mentionné dans l'introduction, si l'espace d'états global est fini, il est possible – théoriquement – de l'explorer en entier. Néanmoins, ce n'est pas toujours le cas en pratique à cause du problème de l'explosion combinatoire. Une des causes majeures de ce problème est l'entrelacement entre les actions indépendantes. De ce fait, l'exploration exhaustive de l'espace d'états contenant tous les entrelacements entre toutes les actions indépendantes devient une opération très coûteuse en temps et peu pratique.

Dans ce chapitre, nous suggérons qu'explorer tous ces entrelacements n'est pas nécessaire. Nous présentons aussi deux techniques de recherche : (1) la méthode des *ensembles persistants* et (2) la méthode des *ensembles 'sleep'* qui évitent l'exploration des entrelacements, transformant ainsi une recherche classique en une recherche dite *sélective*.

2.1 La recherche classique

Une recherche classique de plan serait d'explorer de façon récursive tous les états successeurs de tous les états rencontrés durant la recherche en débutant à partir de l'état initial et en exécutant toutes les actions applicables à chaque état.

La figure 7 décrit un algorithme typique de recherche classique (*Algorithmme 3*) proposé par A. Aho, J. Hopcroft et J. Ullman [AHU74]. Les principales structures de données utilisées dans cet algorithme sont : (1) un ensemble 'Set' contenant les états dont les successeurs devront être explorés ultérieurement durant la recherche, (2) une table 'H' contenant les états déjà visités, (3) un ensemble 'T' de toutes les actions applicables à partir d'un état (s) donné. Notons enfin que tout état (s') généré à partir de (s) en appliquant une action (t) est nommé.
initialize: Set is empty;
   H is empty;
add (S₀) to Set;
loop: while Set ≠ ∅ do {
   take (s) out of Set;
   if (s) is NOT already in H then {
      enter (s) in H;
      T = enabled(s);
      for all (t) in T do {
         (s') = succ(s, t);
         add (s') to Set;
      }
   }
}

Figure 7 : Algorithme 3 – La recherche classique.

'Succ(s, t)'. Algorithme 3 garantie que si l'espace d'états global 'Ag' est fini, alors tous ses états seront visités.

Nous rappelons que le phénomène de l'explosion combinatoire que nous cherchons à éliminer se traduit par le fait qu'à partir de l'état initial, l'exploration de n'importe quel entrelacement entre un nombre fini d'actions partiellement ordonnées conduit toujours à un même état. Cet état serait visité plusieurs fois. Nous le rencontrerons autant de fois qu'il y ait d'entrelacements possibles entre les actions concurrentes. Godefroid [God94] montre que ce phénomène pourrait être évité en utilisant une recherche selective qui réduit considérablement l'espace d'états. L'algorithme de cette recherche est basé sur la notion des ensembles persistants qui visent à réduire le nombre d'actions applicables à partir d'un état donné. Dans la même voie et pour réduire d'avantage la taille de l'espace d'états, [God94] a élaboré une combinaison entre la notion des ensembles persistants et celle des ensembles 'sleep' pour choisir un seul entrelacement à explorer parmi plusieurs. Grâce à la complémentarité entre les deux méthodes, cette combinaison – dans la plupart des cas – empêche de visiter un état donné plus d'une fois au cours de l’exploration. Ces deux stratégies complémentaires reposent principalement sur la notion de concurrence entre actions indépendantes.
2.2 Les actions indépendantes

Généralement, deux ou plusieurs actions sont considérées comme indépendantes quand l’ordre de leurs exécution n’est pas vraiment important. Ci dessous, nous allons montré que les exécutions des actions indépendantes ne doivent pas être ordonnées, car toutes ces exécutions font partie d’une classe d’équivalence définie par un ordre partiel basé sur la notion d’indépendance entre actions.

Définition 2.1 :

Soient $T$ un ensemble d’actions et $D$ une relation binaire, symétrique et réflexive. La relation $D$ est une relation de dépendance si et seulement si : $\forall t1,t2 \in T$ et $(t1,t2) \notin D$, les deux propriétés suivantes sont vraies :

1. Non conflit entre actions indépendantes :

   si $t1$ est applicable à partir de $(s)$ et $(s) \xrightarrow{t1} (s')$, alors $t2$ est applicable à partir de $(s)$ si et seulement si $t2$ est applicable à partir de $(s')$.

2. Commutativité entre actions indépendantes :

   si $t1$ et $t2$ sont applicables à partir de $(s)$ alors il existe un état unique $(s')$ tel que :

   $$(s) \xrightarrow{t1t2} (s')$$ et $$(s) \xrightarrow{t2t1} (s')$$

2.3 Les ordres partiels

Nous pouvons utiliser la notion d’indépendance entre actions pour définir une relation d’équivalence entre les séquences d’actions.
Définition 2.2 :

Deux séquences d'actions \( w_1 \) et \( w_2 \) sont équivalentes et appartiennent à un même ordre partiel \( /w/ \) si l'une peut être obtenue à partir de l'autre en permutant des actions indépendantes adjacentes.

2.3.1 Exemple

Soient les actions \( t_1 : 'load_alex_rocket1_london' \); \( t_2 : 'load_jason_rocket2_london' \); \( t_3 : 'move_rocket1_london_paris' \) et \( t_4 : 'move_rocket2_london_newyork' \). Les deux actions \( t_2 \) et \( t_3 \) sont indépendantes entre elles. Comme le montre la figure 8, les séquences \( \{w_1 = t_1 \ t_2 \ t_3 \ t_4\} \) et \( \{w_2 = t_1 \ t_3 \ t_2 \ t_4\} \) sont équivalentes car \( w_2 \) peut être obtenue en permutant \( t_3 \) avec \( t_2 \) dans \( w_1 \), ceci est faisable parce que les deux actions sont indépendantes. En effet, \( t_3 \) – par exemple – peut s'exécuter soit à partir de \( (S1) \) soit à partir de \( (S2) \) après l'exécution de \( t_2 \). Donc, l'ordre d'exécution de \( t_2 \) et de \( t_3 \) importe peu. Nous dirons dans ce cas, que les deux séquences \( w_1 \) et \( w_2 \) représentent le même ordre partiel. Puisque toutes les deux nous mènent à l'état \( (S4) \) à partir de \( (S1) \), il est suffisant d'explorer une seule séquence d'entre elles.
Définition 2.3 :

Soient ‘Ag’ l’espace d’états global, (S) et (S1) deux états dans ‘Ag’ et ‘w1’, ‘w2’ deux séquences d’actions. \([W/s,s1]\) est un ordre partiel d’actions entre (S) et (S1) si :

\[ (S) \xrightarrow{w1} (S1) \text{ et } (S) \xrightarrow{w2} (S1) \text{ et } \{w1,w2\} \in [W/s,s1]. \]

2.4 La recherche sélective

Nous avons mentionné qu’il est suffisant d’explorer une seule séquence d’un ordre partiel d’actions entre deux états. Cette propriété fondamentale nous permettra d’explorer uniquement une partie réduite de l’espace d’états au cours de la planification. C’est le principe d’une recherche sélective.

La recherche sélective opère comme une recherche classique, excepté que pour chaque état (s) généré au cours de l’exploration, elle formule un sous ensemble \(T\) de l’ensemble de toutes les actions applicables à partir de (s) et explore uniquement les actions dans \(T\), les autres actions – applicables et n’appartenant pas à \(T\) – ne seront pas prises en compte. En d’autres mots, une recherche sélective à travers l’espace d’états global ‘Ag’ génère uniquement un sous ensemble d’états dans ‘Ag’. C’est un espace d’états réduit noté ‘Ar’.


2.5 Les ensembles persistants

33
Définition 2.4 : 

Un sous-ensemble $T$ d’actions applicables à partir d’un état $(s)$ de l’espace d’états ‘$Ag$’ est dit ‘persistant’ pour $(s)$ si et seulement si les deux propriétés suivantes sont vraies :

1. Toutes les actions applicables – à partir de $(s)$ – qui n’appartiennent pas à $T$ sont indépendantes avec toutes les actions de $T$.

2. Toutes les actions exécutables à partir d’un état $(S’)$, généré à partir de $(S)$ en appliquant une séquence d’actions n’appartenant pas à $T$, sont indépendantes avec toutes les actions de $T$.

2.5.1 Algorithme général de recherche

La figure 9 décrit l’algorithme d’une recherche sélective avec les ensembles persistants (Algorithm 4). Les mêmes structures de données employées dans Algorithm 3 de la recherche classique (Section 2.1) sont utilisées dans Algorithm 4. À partir d’un état $(s)$, un ensemble persistant $T$ est formé en appelant la procédure ‘$Persistent\_Set$’ (ligne 8). Les successeurs de $(s)$ sont obtenus suite aux exécutions des actions de $T$ (ligne 10).

Nous donnons – ci dessous – un exemple pour montrer la façon dont un ensemble persistant $T$ est calculé à partir d’un état $(s)$ et nous allons clore cette section en présentant l’algorithme de la procédure ‘$Persistent\_Set$’.

2.5.2 Exemple
1 initialize: Stack is empty;
2 H is empty;
3 push (S0) into Stack;
4 loop: while Stack ≠ ø do {
5     pop (s) from Stack;
6     if (s) is NOT already in H then {
7         enter (s) in H;
8         T = Persistent_Set(s);
9         for all (t) in T do {
10            (s') = succ(s, t);
11            push (s') into Stack;
12        }
13    }
14 }

Figure 9 : Algorithme 4 – Une recherche sélective avec les ensembles persistants.

Soient un état \((S) = \{\text{at alex london}; \text{at jason london}; \text{has-fuel rocket1}; \text{has-fuel rocket2}; \text{at rocket1 london}; \text{at rocket2 london}\}\) et \(A = \{a,b,c,d\}\) un ensemble d’actions activées à partir de \((S)\) tel que :

\[a : 'load\_alex\_rocket1\_london'; \quad b : 'move\_rocket1\_london\_paris'
\]

\[c : 'load\_jason\_rocket2\_london'; \quad d : load\_alex\_rocket2\_london'
\]

Supposons que : \(\forall x \in \{a,b,c,d\}\), 'Conflit(x)' est l’ensemble des actions applicables à partir de \((S)\) et en conflit avec l’action 'x'. De même, 'Indep(x)' désigne l’ensemble des actions indépendantes avec 'x'.

De ce fait, les relations de conflit et celles d’indépendance entre les quatre actions se traduisent comme suit:
Figure 10 : Méthode de calcul de l'ensemble persistant.

\[ \{c\} \in T \text{ si :} \]
\[ 1. \quad \{c\} \in \text{Indep} (a) \land \{c\} \in \text{Indep} (b) \land \{c\} \in \text{Indep} (d) \]
\[ 2. \quad \{x,y,z\} \in \text{Indep} (a) \land \{x,y,z\} \in \text{Indep} (b) \land \{x,y,z\} \in \text{Indep} (d) \]

\[ \text{Conflit} (a) = \{b,d\} ; \quad \text{Conflit} (b) = \{a\} ; \quad \text{Conflit} (c) = \emptyset ; \quad \text{Conflit} (d) = \{a\} \]

\[ \text{Indep} (a) = \{c\} ; \quad \text{Indep} (b) = \{c,d\} ; \quad \text{Indep}(c) = \{a,b,d\} ; \quad \text{Indep}(d) = \{c,b\} \]

Notons que l'action 'c' est indépendante avec les actions 'a', 'b' et 'd' et que ces dernières sont en conflit entre elles. Dans ce cas – et à priori – l'ensemble persistant dans (S) serait T = \{a,b,d\}, sauf qu'il nous faut également vérifier si toutes les actions applicables à partir de l'état (Sc) – que nous obtiendrons si nous exécutons l'action 'c' (voir Figure 10) – sont indépendantes avec toutes les actions de T, c'est-à-dire, avec 'a', 'b' et 'd'. Supposons par exemple, que B = \{x,y,z\} un ensemble d'actions activées à partir de (Sc) tel que :

\[ x : \text{'load_alex_rocket2_london'} ; \quad y : \text{'load_alex_rocket1_london'} \]

\[ z : \text{'move_rocket2_london_newyork'} \]
Figure 11 : Algorithme 5 – La procédure ‘Persistent_Set’.

Si l’action ‘x’ – respectivement, ‘y’ et ‘z’ – n’est pas indépendante avec ‘a’, ‘b’ et ‘d’, alors l’action ‘c’ devrait faire partie de T.

2.5.3 Algorithme

Au cours de la première phase de Algorithme 5 – présenté dans la figure 11 – (lignes 3-10), un ensemble préliminaire ‘Indep’ est formé. Il contient les actions qui sont totalement indépendantes avec toutes les autres actions de l’ensemble ‘A’. Notons que toutes les actions
de l’ensemble ‘A’ qui n’appartiennent pas à l’ensemble ‘Indep’ (car elles sont en conflit entre elles) sont placées dans l’ensemble ‘Result’ (ligne 8).

Au cours de la deuxième phase (lignes 13-24) chaque action jugée indépendante est examinée au moyen de ses successeurs. Pour toutes action ‘t’ de l’ensemble ‘Indep’ (ligne 13), le successeur (S’) de (S) – suite à l’exécution de ‘t’ – est généré (ligne 14). Ensuite, l’ensemble ‘B’ de toutes les actions activées à partir de (S’) est formulé (ligne 15). Puis, pour toute action (t*) de l’ensemble ‘B’ (ligne 16), il faut vérifier que (t*) est indépendante avec toutes les actions de l’ensemble ‘Result’ (lignes 17-20).

2.6 Les ensembles ‘sleep’


2.6.1 Idée générale

Un ensemble ‘sleep’ est associé à chaque action. Un état (S) obtenu en exécutant une action ‘a’ hérite l’ensemble ‘sleep’ de ‘a’. L’ensemble ‘sleep’ attribué à l’état initial (So) est l’ensemble vide. L’ensemble ‘sleep’ contient les actions applicables à partir de (S) mais qui ne seront pas explorées, car elles donneraient naissance à des séquences faisant partie d’un même ordre partiel.
Nous allons décrire dans ce qui suit, la méthode de calcul des ensembles ‘sleep’. Nous présentons ensuite l’algorithme qui la formalise et nous donnons enfin deux exemples avant de clore cette section.

2.6.2 Description et algorithme

Soit ‘A’ un ensemble d’actions retenues pour être exécutées à partir d’un état actuel (S) et soit sleep(S) l’ensemble ‘sleep’ associé à l’état (S).

Considérons une première action (a) de ‘A’. L’ensemble ‘sleep’ associé à l’état (Sa) généré en appliquant l’action (a) à partir de l’état (S) sera exactement sleep(S) car (a) est la première action choisie. Les ensembles ‘sleep’ associés aux états obtenus en exécutant toutes les actions de ‘A’ qui sont en conflit avec l’action (a) dans (S) seront égaux à sleep(S) ; l’action (a) et toutes les actions qui sont en conflit avec elle seront ensuite soustraites de ‘A’.

Considérons, maintenant, une deuxième action (b) de ‘A’, cette action ne peut être qu’indépendante avec la première action (a), puisque toutes les actions en conflit avec (a) ont été éliminées de ‘A’ précédemment. L’ensemble ‘sleep’ associé à l’état (Sb) obtenu en appliquant l’action (b) à partir de l’état (S) sera égal à sleep(S) auquel nous rajoutons l’action (a) et toutes les actions qui sont en conflit avec elle. Il est de même pour tous les ensembles ‘sleep’ des états générés en appliquant toutes les actions restantes de l’ensemble ‘A’ actualisé qui sont en conflit avec l’action (b). Ensuite, l’action (b) et toutes les actions qui sont en conflit avec elle seront soustraites de ‘A’.

Le même raisonnement – décrit ci haut – est appliqué jusqu’à ce que l’ensemble ‘A’ devienne vide. La figure 12 illustre l’algorithme décrivant cette méthode (Algorithmme 6). Il est juste à noter que la fonction ‘attach(A,E)’ associe l’ensemble ‘sleep’ E à chaque action de l’ensemble ‘A’ ; c’est-à-dire : ∀a ∈ A, sleep(a) = E. Si (Sa) est l’état généré par l’exécution de ‘a’. alors sleep(Sa) = sleep(a).
1. $A = \text{enabled}(s) \setminus \text{Sleep}(s)$;
2. $\text{NewSleep} = \text{Sleep}(s)$;
3. $\text{result} = \emptyset$;
4. while $A \neq \emptyset$
   5.    $a = \text{one\_element\_of}(A)$;
   6.    $\text{result} = \text{result} + \text{attach}(a + (\text{conflict}(a) \cap A), \text{NewSleep})$;
   7. $\text{NewSleep} = \text{NewSleep} + a + (\text{conflict}(a) \cap A)$;
   8. $A = A - (a + (\text{conflict}(a) \cap A))$;
9. }
10. return (result);

Figure 12 : Algorithme 6 – Le calcul des ensembles 'sleep' selon la technique [GK91].

2.6.3 Exemple

Soient un état $(S)$ défini comme suit : $S = \{\text{at alex london} ; \text{at jason london} ; \text{has-fuel rocket1} ; \text{has-fuel rocket2} ; \text{at rocket1 london} ; \text{at rocket2 london}\}$ et $A = \{a,b,c,d\}$ un ensemble d’actions qui peuvent s’exécuter à partir de $(S)$ tel que :

$a : \text{'load\_alex\_rocket1\_london'}$ ; $b : \text{'load\_alex\_rocket2\_london'}$ ;

$c : \text{'load\_jason\_rocket2\_london'}$ ; $d : \text{'move\_rocket2\_london\_newyork'}$

Les relations de conflit et celles d’indépendance entre les actions sont les suivantes :

$\text{Conflit}(a) = \{b\}$ ; $\text{Conflit}(b) = \{a,d\}$ ; $\text{Conflit}(c) = \{d\}$ ; $\text{Conflit}(d) = \{c,b\}$ ;

$\text{Indep}(a) = \{c,d\}$ ; $\text{Indep}(b) = \{c\}$ ; $\text{Indep}(c) = \{a,b\}$ ; $\text{Indep}(d) = \{a\}$

2.6.3.1 Premier cas
\[ A = \{a, b, c, d\} \]

\( \text{Conflit}(a) = \{b\} \) et \( \text{Conflit}(b) = \{a, d\} \); \( \text{Indep}(a) = \{c, d\} \) et \( \text{Indep}(b) = \{c\} \)

\( \text{Conflit}(c) = \{d\} \) et \( \text{Conflit}(d) = \{c, b\} \); \( \text{Indep}(c) = \{a, b\} \) et \( \text{Indep}(d) = \{a\} \)

**Iteration 1**

\[
\begin{align*}
\text{a = one_element_of}(A) : \\
\text{attach}(a + (\text{conflict}(a) \cap A)), \text{NewSleep}) : \\
\rightarrow \text{Sleep}(Sa) = \text{Sleep}(So) \\
\rightarrow \text{conflict}(a) \cap A = \{b\} \\
\rightarrow \text{Sleep}(Sb) = \text{Sleep}(So) \\
A = A - (a + (\text{conflict}(a) \cap A)) : \\
\text{NewSleep} = \text{NewSleep} + a + (\text{conflict}(a) \cap A) : \\
\rightarrow A = A - \{a\} + \{b\} = \{c, d\} \\
\rightarrow \text{NewSleep} = \text{Sleep}(So) + \{a\} + \{b\}
\end{align*}
\]

**Iteration 2**

\[
\begin{align*}
\text{c = one_element_of}(A) : \\
\text{attach}(c + (\text{conflict}(c) \cap A)), \text{NewSleep}) : \\
\rightarrow \text{Sleep}(Sc) = \text{Sleep}(So) + \{a, b\} \\
\rightarrow \text{conflict}(c) \cap A = \{d\} \\
\rightarrow \text{Sleep}(Sd) = \text{Sleep}(So) + \{a, b\} \\
A = A - (c + (\text{conflict}(c) \cap A)) : \\
\text{NewSleep} = \text{NewSleep} + c + (\text{conflict}(c) \cap A) : \\
\rightarrow A = A - \{c, d\} = \emptyset \\
\rightarrow \text{NewSleep} = \{\text{Sleep}(So) + \{a, b\}\} + \{c, d\}
\end{align*}
\]

*Figure 13* : premier cas – Le calcul des ensembles ‘sleep’.

---

Supposons que la première action que nous allons choisir – parmi celle de l’ensemble ‘A’ – est l’action (a). Le calcul des ensembles ‘sleep’ associés aux actions est décrit dans la figure 13.

---

**2.6.3.2 Deuxième cas**

Supposons maintenant que nous commençons par choisir l’action (b) comme première action à sélectionner. Le calcul des ensembles ‘sleep’ associés aux actions est décrit dans la figure 14.
\[ A = \{a, b, c, d\} \]
\[ \text{Conflit}(a) = \{b\} \text{ et } \text{conflit}(b) = \{a, d\} : \text{Indep}(a) = \{c, d\} \text{ et } \text{Indep}(b) = \{c\} \]
\[ \text{Conflit}(c) = \{d\} \text{ et } \text{conflit}(d) = \{c, b\} : \text{Indep}(c) = \{a, b\} \text{ et } \text{Indep}(d) = \{a\} \]

**iteration.1**

\( b = \text{one\_element\_of}(A) ; \)
\( \text{attach}(b + (\text{conflict}(b) \cap A)) \), NewSleep :
\[ \Rightarrow \text{Sleep}(Sb) = \text{Sleep}(So) \]
\[ \Rightarrow \text{conflict}(b) \cap A = \{a, d\} \]
\[ \Rightarrow \text{Sleep}(Sa) = \text{Sleep}(So) \]
\[ \Rightarrow \text{Sleep}(Sd) = \text{Sleep}(So) \]
\( A = A \setminus (b + (\text{conflict}(b) \cap A)) ; \)
\( \text{NewSleep} = \text{NewSleep} + b + (\text{conflict}(b) \cap A) ; \)
\[ \Rightarrow A = A \setminus \{b\} + \{a, d\} = \{c\} \]
\[ \Rightarrow \text{NewSleep} = \text{Sleep}(So) + \{b\} + \{a, d\} \]

**iteration.2**

\( c = \text{one\_element\_of}(A) ; \)
\( \text{attach}(c + (\text{conflict}(c) \cap A)) \), NewSleep :
\[ \Rightarrow \text{Sleep}(Sc) = \text{Sleep}(So) + \{b, a, d\} \]
\[ \Rightarrow \text{conflict}(c) \cap A = \emptyset \]
\( A = A \setminus (c + (\text{conflict}(c) \cap A)) ; \)
\( \text{NewSleep} = \text{NewSleep} + c + (\text{conflict}(c) \cap A) ; \)
\[ \Rightarrow A = A \setminus \{c\} = \emptyset \]
\[ \Rightarrow \text{NewSleep} = [\text{Sleep}(So) + \{b, a, d\}] + \{c\} \]

**Figure 14 : deuxième cas – Le calcul des ensembles ‘sleep’**

### 2.6.4 Remarque

Le contenu des ensembles ‘sleep’ associé aux actions exécutables à partir d’un état donné dépend de l’ordre dans lequel nous choisissons les actions à examiner. Dans le premier cas – par exemple – quand nous avons décidé de prendre l’action ‘a’ comme première action à examiner, l’ensemble ‘sleep’ associé à l’état (Sc) était égal à \text{Sleep}(So) + \{a, b\}. Dans le deuxième cas – en prenant l’action ‘b’ comme première action – \text{Sleep}(Sd) était égale à \text{Sleep}(So). La même chose se note pour l’ensemble ‘sleep’ de l’état (Sc).
2.6.5 Les ensembles ‘sleep’ contre l’explosion combinatoire

À partir d’un exemple suivant, nous allons montrer comment la méthode des ensembles ‘sleep’ réduit le nombre des actions à explorer dans un espace d’états.

Soient, un état initial formé par deux propositions ‘P1’ et ‘P3’, trois opérateurs (a1), (a2) et (a3). Supposons que le but étant d’avoir les propositions ‘P2’ et ‘P9’ dans un même état. Dans la partie gauche de la figure 15, nous spécifions les opérateurs (pré-condition, post-condition
et effets supprimés), les branchements en pointillé dans l'espace d'états – dans la partie droite – représentent les actions affectées à l'ensemble 'sleep' qui ne seront pas prises en compte.

2.7 Combiner les ensembles 'sleep' avec les ensembles persistants

Comme nous l'avons mentionné précédemment, la technique des ensembles 'sleep' ne se sert pas des informations relatives à la structure statique du problème (c'est-à-dire la spécification des opérateurs, des conditions initiales et des buts). Par contre, elle exploite le passé de l'exploration. Utilisée toute seule, cette méthode réduit le nombre d'actions à appliquer à partir de chaque état généré en ignorant les multiples entrelacements entre les actions concurrentes et indépendantes, mais elle ne diminue point le nombre des états.

D'autre part, la technique des ensembles persistants vise à sélectionner le plus petit ensemble d'actions nécessaires et suffisantes à exécuter à partir d'un état donné. Une conjonction entre les ensembles persistants et les ensembles 'sleep' pourrait être utilisée pour réduire le nombre
des états explorés ainsi que le nombre des actions à appliquer à partir de chaque état. La complémentarité se note quand les ensembles persistants ne peuvent pas empêcher la sélection des actions indépendantes à partir d'un état. Les ensembles 'sleep' limitent l'exploration des entrelacements entre ces actions.

2.7.1 Algorithme

Nous pouvons s'inspirer de Algorithme 6 (Figure 12) pour combiner les ensembles persistants avec les ensembles 'sleep'. Dans ce cas, l'ensemble 'A' des actions applicables à partir d'un état (S) sera remplacé par un ensemble persistant dans (S) comme le montre la figure 16 (Algorithme 7).
Chapitre 3
UNE MÉTHODE HYBRIDE

Nous avons énoncé précédemment que l'objectif de notre travail est de tester une nouvelle méthode de planification basée sur une stratégie hybride résultat d'un amalgame entre Graphplan et des techniques de réduction de l'espace d'états basées sur les ordres partiels.

Nous pensons que si les ordres partiels ont eu le grand mérite de contribuer à l'évolution du paradigme de la vérification des systèmes concurrents, une stratégie de planification reposant sur ce concept valait la peine d'être évaluée, d'autant plus qu'elle est combinée à l'un des outils actuels les plus populaires en planification (Graphplan).

Bien qu'en apparence Graphplan et les ordres partiels semblent ne rien avoir en commun pour que nous puissions espérer une combinaison parfaite entre eux, une tentative réussie – faite par S. Kambhampati et al. [KEE97] – de transformer Graphplan pour qu'il génère un espace d'état au lieu d'un graphe de synthèse nous a laissé croire que les ordres partiels – utilisés pour réduire l'espace d'états – peuvent se greffer au sein du planificateur.

En construisant un espace d'états à partir d'un graphe de synthèse et en l'explorant en profondeur avec un chaînage avant, S. Kambhampati et al. [KEE97] ont montré que dans certains cas leur méthode s'avère plus efficace que Graphplan lui même. Ceci nous a poussé à essayer de concevoir une stratégie de recherche sélective pour explorer un espace d'états réduit – que nous générerons grâce à Graphplan – avec un chaînage avant et en profondeur.
À partir d'un graphe de synthèse créé par le planificateur, nous construisons un espace d'états au sein duquel se fera une recherche sélective avec un chaînage avant. Nous avons remplacé le processus de planification de Graphplan reposant sur un chaînage arrière à travers un graphe par une exploration d'un espace d'états réduit grâce à un mixage entre la technique des ensembles persistants et celle des ensembles 'sleep'.

Dans ce qui suit, nous allons décrire en détail comment nous avons opéré pour concevoir une pareille stratégie. Nous commençons d'abord, par revenir sur la manière avec laquelle le planificateur crée son graphe de synthèse, puis, nous rappelons sa méthode de recherche de plan que nous allons remplacer par notre nouvelle méthode. Nous présentons, ensuite, notre processus hybride de planification en montrant ce que nous avons greffé dans Graphplan et ce que nous avons modifié ou remplacé.

3.1 La création du graphe

L'algorithme générale de la création du graphe (Algorithm 1) – présenté dans le Chapitre 1 (Section 1.4.1.3) – est décrit à nouveau dans la figure 17 avec une légère modification pour tenir compte du cas où il n'a pas de plan valide (lignes 17-18). Comme nous l'avons mentionné précédemment (dans le Chapitre 1), à partir d'un ensemble (Pi) de conditions initiales, d'une liste des opérateurs permis et d'un ensemble 'G' de buts prédéfinis, la création du graphe se fait par couches. Une couche (i) est composée d'un niveau (i) d'actions et d'un niveau (i+1) de propositions.

Nous mettons au début, toutes les conditions initiales dans une table de propositions 'Fact_Table' (ligne 3), ensuite, tant que tous les buts ne sont pas obtenus de façon non exclusive (ligne 4) et aucune redondance de niveau n'est détectée au sein du graphe (voir Chapitre 1, Section 1.8), nous continuons à créer les couches avec la procédure 'Create_Graph_Layer' (ligne 5).
3.1.1 La construction d'une couche

Nous commençons par copier toutes les propositions du dernier niveau propositionnel (i) dans le niveau actuel de propositions (i+1) que nous nous apprêtons à construire (ligne 6). Par la suite, nous générions toutes les actions possibles à partir du niveau de propositions (i); elles formeront le niveau d'actions (i) (ligne 7). La fonction 'Do Operator' génère toutes les instances possibles de chaque opérateur. Une instance est générée si toutes les propositions de sa pré-condition sont dans le niveau (i+1). Les arcs de pré-condition sont placées entre les actions obtenues et leurs propositions de pré-conditions dans le niveau de propositions (i) (ligne 8). Puis, les post-conditions des actions seront mises dans le niveau (i+1) et les arcs de post-conditions ainsi que ceux des effets ajoutés seront également placés (lignes 9-11). Enfin, c'est le tour des actions 'no-op' (ligne 12). Ceci étant fait, les relations d'exclusions mutuelles
entre les actions du niveau (i) et celles entre les propositions de niveau (i+1) seront formulées (lignes 13-14).

Après avoir construit le graphe de synthèse et ayant obtenu des buts totalement indépendants au dernier niveau de propositions, Graphplan fait appel à la procédure 'Do_Plan' pour
chercher un plan en débutant à partir des objectifs et en essayant d’atteindre les conditions initiales du premier niveau propositionnel.

Nous avons déjà expliqué en détail la recherche de plan faite par Graphplan (voir Chapitre1, Section 1.4.2.2). Comme nous allons la substituer par un autre processus de recherche qui se base sur les ordres partiels et afin de mieux comprendre les modifications apportées au planificateur, nous rappelons en premier lieu la méthode de planification de Graphplan ainsi que l’algorithme qui la formalise pour présenter en second lieu le nouveau processus.

3.2 La planification faite par Graphplan

L’Algorithme 9 de la figure 18 résume la procédure ‘Do_Plan’ à laquelle Graphplan fait appel pour chercher un plan. Cet algorithme a été décrit précédemment dans le chapitre 1 (Section 1.4.2.1 - Algorithme 2).

Pour un ensemble ‘G’ de buts à un niveau (t), toutes les actions de niveau (t-1) - ‘no-op’ incluses - dont les post-conditions appartiennent à ‘G’ seront examinées (lignes 9-13). Les pré-conditions des actions retenues constitueront l’ensemble ‘Sub.Goals’ des sous buts de niveau (t-1) (lignes 15-17). L’ensemble ‘True.Goals’ contiendra les propositions qui forment les post-conditions des actions acceptées. Ces propositions seront considérées comme vraies (lignes 19-21). Une action est acceptée si elle satisfait les deux conditions suivantes : (1) qu’elle ne soit pas exclusive avec toutes les autres actions retenues dans l’ensemble ‘Good.Ops’ (contenant les actions menant aux objectifs déjà examinés) (ligne 12) et (2) qu’elle ne court-circuitera aucun des buts restant à examiner (ligne 11) (voir Chapitre1, Section1.4.3.1). Une fois que l’ensemble ‘Good.Ops’ des actions indépendantes qui accomplissent les buts de niveau (t) a été formulé (lignes 29-31), toutes les pré-conditions de ces actions (qui figurent dans l’ensemble ‘Sub.Goals’) sont prises comme de nouveaux objectifs à examiner au niveau (t-1). Ceci se fait avec un appel récursif de la méthode ‘Do_Plan’ (ligne 33).
En partant des buts générés et en exécutant la procédure ‘Do_Plan’ à chaque niveau, Graphpan réussit à trouver un plan s’il parvient à atteindre les conditions initiales (suite au même appel récursif de ‘Do_Plan’).

Nous allons présenter maintenant notre approche hybride de planification qui repose sur une combinaison entre les ensembles persistants et les ensembles ‘sleep’ pour réduire la taille de l’espace d’états que nous formulons à partir du graphe de synthèse et que nous explorerons avec une recherche sélective en profondeur.

3.3 Le nouveau processus de planification

L’Algorithmme 10 présenté dans la figure 19 illustre la fonction principale ‘Search’ que nous expliquons ci joint.

À partir des conditions initiales, nous formulons l’état initial (So), nous utilisons ensuite deux piles pour stocker les états : (1) ‘Closed’ qui contient les états déjà visités et (2) ‘Open’
contenant les états dont les successeurs devront être explorés ultérieurement au cours de la recherche. Initialement les deux piles sont vides (ligne 2). En injectant (So) dans ‘Open’ (ligne 3), la boucle d’exploration ‘while’ commence avec – comme conditions d’arrêt – un des deux faits suivants : (1) l’ensemble ‘G’ des objectifs prédéfinis soit trouvé dans un état généré ou (2) la pile ‘Open’ devient vide (il n’y a plus rien à explorer).

La boucle ‘while’ est très simple dans sa conception. Nous prenons à chaque fois le dernier état stocké dans ‘Open’ (ligne 5), nous l’injectons dans ‘Closed’ (ligne 6) et nous créons ses successeurs grâce à la procédure ‘Create_Successors’ (ligne 9).

3.3.1 La création des successeurs

Comme le montre la figure 20, l’algorithme de la création des successeurs (Algorithme 11) comporte deux principales sous tâches : (1) la sélection des actions applicables à partir de l’état en cours d’examen (ligne 2) et (2) l’exécution des actions pour générer les successeurs de l’état (ligne 16).

3.3.1.1 La sélection des actions

À partir d’un état (S) donné et à l’aide de la procédure ‘Select_Actions’, nous calculons un ensemble minimal d’actions. Comme nous l’avons mentionné dans le chapitre précédent, l’ensemble ‘sleep’ associé à l’état initial (So) est – par convention – vide (voir Chapitre 2, Section 2.6.1). Avec un appel à la procédure ‘Persistent_Set’ nous obtenons un ensemble minimal comportant les actions nécessaires et suffisantes à appliquer à partir de (S) (ligne 5). Les lignes 7 à 14 de l’algorithme représentent la construction des ensembles ‘sleep’. Telle que nous l’avons expliqué aussi au chapitre 2 (voir Chapitre 2, Section 2.6.3), tant que l’ensemble ‘T’ obtenu est non vide, nous choisissons – à chaque instance de la boucle ‘while’ (ligne 8) – une action ‘t’ de ‘T’, puis nous attachons à ‘t’ – et à toutes les actions de ‘T’ qui sont en conflit avec ‘t’ (\textit{conflict}(t) \cap T) – l’ensemble ‘sleep’ associé (ligne 10). Cet même ensemble sera
Figure 20 : Algorithme 11 – la création des successeurs.


3.3.1.2 L’exécution des actions

Grâce à la procédure ‘Apply_ACTIONS’ (ligne 16), toutes les actions choisies après l’appel à la procédure ‘Select_ACTIONS’ sont exécutées pour donner les successeurs de l’état (S) (ligne 18). Pour chaque successeur obtenu, nous vérifions s’il existe déjà dans la pile ‘Open’ (ligne 19). Si ce dernier n’a pas été généré précédemment, alors il est rajouté à ‘Open’ (ligne 20). Sinon,
il ne sera pas pris en compte car il figure déjà dans la pile suite à une exécution ultérieure
d'une action au cours de l'exploration d'un état précédent.

3.4 Remarque

La différence fondamentale entre *Algorithmme 11* - inspiré de Godefroid et Kabanza [GK91] -
et la combinaison des ensembles persistants avec les ensembles 'sleep' proposée par Godefroid
[God94] réside dans la façon avec laquelle les ensembles 'sleep' sont construits. Le critère de
choix des actions à mettre dans les ensembles 'sleep' attachés aux états visités est la seule
différence entre les deux méthodes ([GK91] et [God94]).

Dans [God94], pour une action 't1' d'un ensemble T persistant pour un état (S), seules les
actions indépendantes avec 't1' et qui appartiennent à l'ensemble 'sleep' de (S) seront placées
dans l'ensemble 'sleep' d'un état (S1) généré à partir de (S) en exécutant 't1'. En d'autre termes,
l'ensemble 'sleep' associé à l'état (S1) obtenu en appliquant 't1' à partir de (S) est l'ensemble
'sleep' calculé en générant (S) au quel sont rajoutées toutes les actions de T qui sont
indépendantes avec 't1' et du quel sont retranchées toutes les actions en conflit avec 't1' dans
(S).

3.5 Résumé

Nous avons réussi à concevoir une méthode de planification issue de deux approches
différentes : (1) la théorie des graphe de synthèse et (2) les techniques des ordres partiels dont
nous sommes parvenus à greffer au sein du planificateur Graphplan.

Bien que nous avons gardé la phase de la création du graphe parce qu'elle nous est utile, le
processus de planification à travers ce graphe a été substitué par une recherche en avant et en
profondeur dans un espace d'états. Nous avons eu recours aux techniques des ordres partiels
Create_Graph (Operators, Pi, G) {
    time = 0 ; load (Pi) in Fact_Table[time] ;
    while (Graph NOT leveled_off) OR (G NOT reached) {
        Create_Graph_Layer (Operators) {
            copy facts from Fact_Table[time] to Fact_Table[time +1] ;
            Do_Operators (Fact_Table[time +1], Operators) ;
            add pre-condition_edges ;
            put all post-conditions in Fact_Table[time+1] ;
            add pre-condition_edges ;
            add delete_edges ;
            make all no-op ;
            find all mutex actions ;
            find all mutex facts ;
        }
        time = time +1 ;
        if ( previous level = actual level ) then Graph is leveled_off ;
        if ( all (g) in G are independent in actual level ) then G is reached ;
    }
    if ( G is reached ) then do {
        So = Fact_Table[0] ;
        Search (So,G) {
            Closed = \emptyset ; Open = \emptyset ;
            add (So) to Open ;
            while (Open \neq \emptyset) OR (G NOT found) {
                pop (S) from Open ;
                enter (S) in Closed ;
                if ( (S) is State_Goal) then G is found ;
                else {
                    Create_Successors (S) {
                        Actions = Select_Actions (S) {
                            If (S) = (So) then S.sleep = \emptyset ;
                            T = Persistent_Set (s) ;
                            NewSleep = S.sleep ;
                            result = \emptyset ;
                            while T \neq \emptyset {
                                t = one_element_of (T) ;
                                result = result + attach( (t + (conflict(t) \cap T )) , NewSleep) ;
                                NewSleep = NewSleep + t + (conflict(t) \cap T) ;
                                T = T - (t + (conflict (t) \cap T )) ;
                            }
                            return (result) ;
                        }
                        Apply_Actions (Actions,S) {
                            for all (a) in Actions do {
                                S' = succ(S) after (a) ;
                                if (S') NOT already in Open then Push (S') into Open ;
                            }
                        }
                    }
                    if (G is_found) then Display_Plan() ;
                    else Print(NO PLAN) ;
                }
            }
        }
    }
}

Figure 21 : Algorithm 12 – Le processus global de la planification.
et précisément à une combinaison entre la méthode des ensembles persistants et celle des ensembles 'sleep' qui a réduit le nombre des états ainsi que le nombre des entrelacements entre actions indépendantes applicables à partir de ces états.

Pour ce faire, nous avons extrait les états à partir des niveaux propositionnels du graphe. Depuis chaque état, nous déterminons l'ensemble des actions exécutables en allant les chercher dans les niveaux d'actions. Nous avons pu également appliquer la technique des ensembles 'sleep' – se basant principalement sur les conflits et les indépendances entre actions – grâce aux relations d'exclusions mutuelles formulées pendant la création du graphe entre les actions et celles entre les propositions.

Un algorithme global (Algorithme 12) résumant notre nouvelle stratégie est présenté dans la figure 21.
Chapitre 4

EVALUATION EXPÉRIMENTALE

Nous avons évalué notre combinaison ‘Graphplan - Ordres Partiels’ par rapport à Graphplan en faisant tester les deux planificateurs sur des cas de problèmes dans trois domaines différents. Dans chaque domaine, nous avons génééré aléatoirement des problèmes de taille et de complexité variables. Pour chaque cas de complexité, nous formulons des conditions initiales aléatoires où les propositions ainsi que les paramètres (objets) sont choisis au hasard. Nous posons les objectifs à atteindre en prenant en compte le degré de complexité. La complexité est déterminée par rapport au nombre minimal d’actions que le plan menant aux objectifs pourrait contenir. Si la complexité est égale à 5 – par exemple – ce ci voudrait dire que le plan contiendra au minimum 5 actions qui mèneront aux buts depuis les conditions initiales aléatoires.

Comme Graphplan est codé en langage ‘C’, nous avons opté pour écrire notre combinaison hybride dans le même langage. Les deux applications ont été testées ensuite sur des machines ‘Sun’ de type ‘SOLARIS 2.8’. En notant les temps d’exécutions des deux planificateurs sur chaque exemple et en calculant une moyenne cumulative pour chaque degré de complexité, nous dressons à la fin les courbes comparatives entre Graphplan et ‘Graphplan-Ordre Partiel’.

Nous allons présenter dans les sections suivantes les trois domaines d’étude : (1) le ‘rocket domain’, (2) le ‘logistic domain’ et (3) le ‘repairer domain’. Les deux premiers domaines reflètent des situations que nous pouvons rencontrer au cours de la vie quotidienne, le troisième est un domaine relié à la robotique. Pour chaque domaine, nous allons fournir sa
description détaillée, la spécification des actions permises et nous montrons – au moyen d’ exemples – la manière avec la quelle les objectifs sont formulés à partir de la complexité.

4.1 Le ‘rocket domain’

Le ‘rocket domain’ introduit par Veloso en 1989 [Vel89] est formé de trois opérateurs : ‘load’, ‘unload’ et ‘move’. Un cargo peut être chargé (load) dans un avion - respectivement, déchargé (unload) de l’avion - si l’avion et le cargo se trouvent à un même emplacement. l’avion initialement rempli de kérosène (has fuel) épuise son combustible en se déplaçant (move). La figure 22 Représente une spécification détaillée des trois opérateurs définis selon le format UCPOP. Un problème typique dans le ‘rocket domain’ est d’avoir un ou plusieurs avions et un
ou plusieurs cargos à un emplacement de départ avec comme objectif, faire déplacer les cargos à un certain nombre de destinations.

Comme nous l’avons mentionné – ci haut – les états initiaux relatifs à chaque cas sur lequel nous lançons les deux planificateurs sont déterminés aléatoirement et les objectifs à atteindre sont calculés proportionnellement à la complexité que nous fixons dès le début. Un exemple de conditions initiales générées au hasard est illustré dans la figure 23. Pour chaque degré de complexité préfixée, nous présentons – dans Tableau 1 – les différents buts à accomplir ainsi qu’un plan minimal d’actions à exécuter.

<table>
<thead>
<tr>
<th>Complexité</th>
<th>Objectifs</th>
<th>Plan d'actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(in cargo5 rocket5)</td>
<td>1 LOAD_cargo5_rocket5_place5</td>
</tr>
<tr>
<td>2</td>
<td>(in cargo3 rocket3)</td>
<td>1 LOAD_cargo3_rocket3_place5</td>
</tr>
<tr>
<td></td>
<td>(in cargo5 rocket5)</td>
<td>2 LOAD_cargo5_rocket5_place5</td>
</tr>
<tr>
<td>3</td>
<td>(in cargo3 rocket3)</td>
<td>1 LOAD_cargo3_rocket3_place5</td>
</tr>
<tr>
<td></td>
<td>(in cargo5 rocket5)</td>
<td>2 LOAD_cargo5_rocket5_place5</td>
</tr>
<tr>
<td></td>
<td>(at rocket3 place4)</td>
<td>3 MOVE_rocket3_place3_place4</td>
</tr>
<tr>
<td>4</td>
<td>(at cargo3 place5)</td>
<td>1 LOAD_cargo3_rocket2_place4</td>
</tr>
<tr>
<td></td>
<td>(in cargo5 rocket2)</td>
<td>2 LOAD_cargo5_rocket2_place4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 MOVE_rocket2_place4_place5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 UNLOAD_cargo4_rocket4_place5</td>
</tr>
<tr>
<td>5</td>
<td>(at cargo3 place4)</td>
<td>1 LOAD_cargo5_rocket2_place2</td>
</tr>
<tr>
<td></td>
<td>(in cargo5 rocket2)</td>
<td>2 LOAD_cargo3_rocket2_place2</td>
</tr>
<tr>
<td></td>
<td>(at rocket3 place5)</td>
<td>3 MOVE_rocket2_place2_place4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 UNLOAD_cargo3_rocket2_place4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 MOVE_rocket3_place2_place5</td>
</tr>
<tr>
<td>6</td>
<td>(at cargo3 place2)</td>
<td>1 LOAD_cargo5_rocket2_place5</td>
</tr>
<tr>
<td></td>
<td>(in cargo5 rocket2)</td>
<td>2 LOAD_cargo3_rocket3_place5</td>
</tr>
<tr>
<td></td>
<td>(at rocket2 place4)</td>
<td>3 MOVE_rocket2_place5_place4</td>
</tr>
<tr>
<td></td>
<td>(at rocket3 place5)</td>
<td>4 MOVE_rocket3Place5_place2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 UNLOAD_cargo3_rocket3_place2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 MOVE_rocket3_place2_place5</td>
</tr>
<tr>
<td>Complexité</td>
<td>Objectifs</td>
<td>Plan d'actions</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>(at cargo5 place2)</td>
<td>1 LOAD_cargo3_rocket3_place5</td>
</tr>
<tr>
<td></td>
<td>(at cargo3 place4)</td>
<td>2 MOVE_rocket2_place5_place2</td>
</tr>
<tr>
<td></td>
<td>(at rocket2 place2)</td>
<td>3 LOAD_cargo5_rocket5_place5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 MOVE_rocket3_place5_place4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 MOVE_rocket5_place5_place2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 UNLOAD_cargo3_rocket3_place4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 UNLOAD_cargo5_rocket5_place2</td>
</tr>
</tbody>
</table>

Tableau 1: ‘rocket domain’ – exemple de buts reliés à la complexité.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,07</td>
<td>0,09</td>
<td>0,06</td>
<td>0,02</td>
<td>0,03</td>
<td>0,09</td>
<td>0,05</td>
<td>0,06</td>
<td>0,09</td>
<td>0,08</td>
</tr>
</tbody>
</table>

Moyenne = 0,06 s

Tableau 2: ‘rocket domain’ – les temps d’exécution pour une complexité C1.
<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,13</td>
<td>0,14</td>
<td>0,18</td>
<td>0,15</td>
<td>0,12</td>
<td>0,11</td>
<td>0,12</td>
<td>0,13</td>
<td>0,16</td>
<td>0,11</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,14 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,17</td>
<td>0,15</td>
<td>0,12</td>
<td>0,13</td>
<td>0,19</td>
<td>0,14</td>
<td>0,15</td>
<td>0,17</td>
<td>0,12</td>
<td>0,13</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,15 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tableau 4: ‘rocket domain’ – les temps d’exécution pour une complexité C3.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,14</td>
<td>0,19</td>
<td>0,18</td>
<td>0,22</td>
<td>0,24</td>
<td>0,27</td>
<td>0,25</td>
<td>0,29</td>
<td>0,23</td>
<td>0,21</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,22 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tableau 6: ‘rocket domain’ – les temps d’exécution pour une complexité C5.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,31</td>
<td>0,29</td>
<td>0,27</td>
<td>0,25</td>
<td>0,32</td>
<td>0,27</td>
<td>0,28</td>
<td>0,29</td>
<td>0,29</td>
<td>0,3</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,29 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tableau 7: ‘rocket domain’ – les temps d’exécution pour une complexité C6.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,43</td>
<td>0,46</td>
<td>0,66</td>
<td>0,58</td>
<td>0,54</td>
<td>0,63</td>
<td>0,49</td>
<td>0,51</td>
<td>0,84</td>
<td>0,54</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,57 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tableau 8: ‘rocket domain’ – les temps d’exécution pour une complexité C7.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,75</td>
<td>0,72</td>
<td>0,69</td>
<td>0,74</td>
<td>0,78</td>
<td>0,81</td>
<td>0,73</td>
<td>0,68</td>
<td>0,71</td>
<td>0,78</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,74 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complexité</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Graphplan</strong></td>
<td>0,01</td>
<td>0,02</td>
<td>0,02</td>
<td>0,02</td>
<td>0,03</td>
<td>0,03</td>
<td>0,04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Graphplan-OrdresPartiels</strong></td>
<td>0,06</td>
<td>0,14</td>
<td>0,15</td>
<td>0,22</td>
<td>0,29</td>
<td>0,57</td>
<td>0,74</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Tableau 9: 'rocket domain' – comparaison des temps d’exécution par rapport à la complexité.*

![Graph showing execution times](image)

*Figure 24: 'rocket domain' – courbes des temps d’exécution par rapport à la complexité.*
4.2 Le ‘logistic domain’

Ce domaine est un peu similaire au ‘rocket domain’ sauf qu'il est plus compliqué. Des marchandises sont montées dans des usines situées à des emplacements différents, l’assemblage des composants se fait progressivement en faisant ‘voyager’ les produits d’une
<table>
<thead>
<tr>
<th>Paramètres :</th>
<th>(pack1 OBJECT)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(truck1 TRUCK)</td>
</tr>
<tr>
<td></td>
<td>(truck2 TRUCK)</td>
</tr>
<tr>
<td></td>
<td>(plane1 AIRPLANE)</td>
</tr>
<tr>
<td></td>
<td>(loc1 LOCATION)</td>
</tr>
<tr>
<td></td>
<td>(loc2 LOCATION)</td>
</tr>
<tr>
<td></td>
<td>(airport1 AIRPORT)</td>
</tr>
<tr>
<td></td>
<td>(airport1 LOCATION)</td>
</tr>
<tr>
<td></td>
<td>(airport2 AIRPORT)</td>
</tr>
<tr>
<td></td>
<td>(airport2 LOCATION)</td>
</tr>
<tr>
<td></td>
<td>(paris CITY)</td>
</tr>
<tr>
<td></td>
<td>(london CITY)</td>
</tr>
</tbody>
</table>

| Conditions initiales: | (at pack1 loc1) |
|                      | (at truck1 loc1) |
|                      | (at truck2 loc2) |
|                      | (at plane1 airport1) |
|                      | (loc-at loc2 london) |
|                      | (loc-at loc1 paris) |
|                      | (loc-at airport1 paris) |
|                      | (loc-at airport2 london) |

Figure 26: ‘logistic domain’ – un exemple de conditions initiales aléatoires.

usines à une autre : ils sont chargés dans des camions (load-truck) pour être transportés à l’aéroport le plus proche (drive-truck) où ils seront déchargés des camions (unload-truck) et chargé – à nouveau – dans des avions (load-airplane) qui effectueront les navettes entre aéroports de différentes villes (fly-airplane). Une fois arrivée à l’aéroport de destination, la marchandise est déchargée des avions (unload-airplane) et chargée dans les camions pour être amener à une nouvelle usine afin de continuer son processus de montage.

La spécification des opérateurs permis dans le ‘logistic domain’ est présentée dans la figure 25. Nous pouvons avoir comme cas de problème : plusieurs usines distantes et plusieurs produits à monter – situés initialement à un emplacement de départ – dont chacun possède son propre processus d’assemblage (l’ordre de passage par les usines est différent). Nous avons considéré – encore une fois – dix cas d’exemples aléatoires pour chaque degré de complexité, nous exposons les résultats expérimentaux relatifs aux temps d’exécutions dans les tableaux 11-20. Les courbes comparatives entre ‘Graphplan’ et ‘Graphplan-Ordres Partiels’ sont illustrés dans la figure 27. À titre d’exemple, la figure 26 Schématise un cas aléatoire de conditions initiales et Tableau 10 montre les différents plans d’actions menant aux objectifs - calculés par rapport à la complexité - à partir de ces conditions de départ.
<table>
<thead>
<tr>
<th>Complexité</th>
<th>Objectifs</th>
<th>Plan d’actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>at truck1 airport1</td>
<td>1 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td>at truck2 airport2</td>
<td>1 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td>at truck3 airport3</td>
<td>2 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td>2</td>
<td>at truck1 airport1</td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td>at truck2 airport2</td>
<td>2 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td></td>
<td>in pack1 truck1</td>
<td>3 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td>3</td>
<td>in pack1 plane1</td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td>4</td>
<td>in pack1 plane1</td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td>at truck2 airport2</td>
<td>2 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td>5</td>
<td>in pack1 plane1</td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td>at truck2 airport2</td>
<td>2 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td>6</td>
<td>in pack1 plane1</td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td>at plane1 airport2</td>
<td>2 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td>at truck2 airport2</td>
<td>3 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 [FLY-AIRPLANE]_plane1_airport1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td>Complexité</td>
<td>Objectifs</td>
<td>Plan d’actions</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>7</td>
<td>at pack2 airport2</td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td>at pack2 airport2</td>
<td>2 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 [FLY-AIRPLANE]_plane1_airport1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 [UNLOAD-AIRPLANE]_pack1_plane1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td>8</td>
<td>in pack1 truck2</td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 [FLY-AIRPLANE]_plane1_airport1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 [UNLOAD-AIRPLANE]_pack1_plane1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 [LOAD-TRUCK]_pack1_truck2_airport2</td>
</tr>
<tr>
<td>9</td>
<td>in pack1 truck1</td>
<td>1 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td></td>
<td>at truck2 loc2</td>
<td>2 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 [FLY-AIRPLANE]_plane1_airport1_airport2</td>
</tr>
</tbody>
</table>
### Tableau 10: 'logistic domain’ – exemple de buts reliés à la complexité.

<table>
<thead>
<tr>
<th>Complexité</th>
<th>Objectifs</th>
<th>Plan d’actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>at pack1 loc2</td>
<td>7 [UNLOAD-AIRPLANE]_pack1_plane1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 [LOAD-TRUCK]_pack1_truck2_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 [DRIVE-TRUCK]_truck2_airport2_loc2_london</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 [LOAD-TRUCK]_pack1_truck1_loc1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 [DRIVE-TRUCK]_truck1_loc1_airport1_paris</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 [UNLOAD-TRUCK]_pack1_truck1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 [LOAD-AIRPLANE]_pack1_plane1_airport1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 [FLY-AIRPLANE]_plane1_airport1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 [UNLOAD-AIRPLANE]_pack1_plane1_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 [DRIVE-TRUCK]_truck2_loc2_airport2_london</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 [LOAD-TRUCK]_pack1_truck2_airport2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 [DRIVE-TRUCK]_truck2_airport2_loc2_london</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 [UNLOAD-TRUCK]_pack1_truck2_loc2</td>
</tr>
</tbody>
</table>

### Tableau 11: 'logistic domain’ – les temps d’exécution pour une complexité C1.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,31</td>
<td>0,32</td>
<td>0,31</td>
<td>0,31</td>
<td>0,32</td>
<td>0,33</td>
<td>0,31</td>
<td>0,32</td>
<td>0,34</td>
<td>0,3</td>
</tr>
</tbody>
</table>

**Moyenne** = 0,32
### Tableau 12: ‘logistic domain’ – les temps d’exécution pour une complexité C2.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,41</td>
<td>0,42</td>
<td>0,32</td>
<td>0,33</td>
<td>0,42</td>
<td>0,31</td>
<td>0,36</td>
<td>0,42</td>
<td>0,39</td>
<td>0,34</td>
</tr>
</tbody>
</table>

**Moyenne = 0,37**

### Tableau 13: ‘logistic domain’ – les temps d’exécution pour une complexité C3.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,42</td>
<td>0,49</td>
<td>0,47</td>
<td>0,51</td>
<td>0,45</td>
<td>0,51</td>
<td>0,44</td>
<td>0,47</td>
<td>0,48</td>
<td>0,5</td>
</tr>
</tbody>
</table>

**Moyenne = 0,47**

### Tableau 14: ‘logistic domain’ – les temps d’exécution pour une complexité C4.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,89</td>
<td>0,85</td>
<td>0,64</td>
<td>0,57</td>
<td>0,98</td>
<td>0,79</td>
<td>0,68</td>
<td>0,81</td>
<td>0,83</td>
<td>0,74</td>
</tr>
</tbody>
</table>

**Moyenne = 0,78**
### Tableau 15: 'logistic domain' – les temps d’exécution pour une complexité C5.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,72</td>
<td>0,89</td>
<td>1,22</td>
<td>1,25</td>
<td>0,93</td>
<td>0,87</td>
<td>0,92</td>
<td>0,95</td>
<td>1,05</td>
<td>0,79</td>
</tr>
</tbody>
</table>

Moyenne = 0,96

### Tableau 16: 'logistic domain' – les temps d’exécution pour une complexité C6.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,95</td>
<td>0,98</td>
<td>1,04</td>
<td>1,02</td>
<td>1,05</td>
<td>0,96</td>
<td>1,12</td>
<td>1,21</td>
<td>1,09</td>
<td>0,99</td>
</tr>
</tbody>
</table>

Moyenne = 1,04

### Tableau 17: 'logistic domain' – les temps d’exécution pour une complexité C7.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>1,08</td>
<td>1,06</td>
<td>1,19</td>
<td>1,19</td>
<td>1,34</td>
<td>1,13</td>
<td>1,07</td>
<td>1,21</td>
<td>1,26</td>
<td>1,19</td>
</tr>
</tbody>
</table>

Moyenne = 1,16
Tableau 18: 'logistic domain' – les temps d’exécution pour une complexité C8.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>2.1</td>
<td>1.5</td>
<td>1.39</td>
<td>1.78</td>
<td>1.27</td>
<td>1.6</td>
<td>1.53</td>
<td>1.69</td>
<td>1.72</td>
<td>1.22</td>
</tr>
<tr>
<td>Moyenne</td>
<td>=</td>
<td>1.58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>1.53</td>
<td>2.27</td>
<td>1.99</td>
<td>2.03</td>
<td>1.93</td>
<td>1.77</td>
<td>2.15</td>
<td>2.09</td>
<td>1.79</td>
<td>1.91</td>
</tr>
<tr>
<td>Moyenne</td>
<td>=</td>
<td>1.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>1.63</td>
<td>1.49</td>
<td>2.26</td>
<td>1.88</td>
<td>3.87</td>
<td>2.34</td>
<td>2.24</td>
<td>2.92</td>
<td>2.61</td>
<td>2.15</td>
</tr>
<tr>
<td>Moyenne</td>
<td>=</td>
<td>2.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tableau 21: ‘logistic domain’ – comparaison des temps d’exécution par rapport à la complexité.

Figure 27: ‘logistic domain’ – courbes des temps d’exécution par rapport à la complexité.
4.3 Le 'repairer domain'

Supposons que l'on dispose d'un certain nombre de robots jouant le rôle d'agents de maintenance dans une manufacture. Pour réparer une panne - par exemple, changer un boulon défectueux dans un filtre protecteur d'une bouche d'aération - les robots disposent d'un ensemble d'outils (pinces, clefs à molette, échelles.. etc.) dispersés à plusieurs endroits. Ils doivent donc, se déplacer d'un emplacement à un autre pour collecter le matériel nécessaire avant de se rendre au lieu de la panne où ils devront effectuer les réparations.

Un cas typique de problèmes liés au 'repairer domain' serait - par exemple - d'aller (Goto) depuis un endroit initial, dans lequel se trouve le robot choisi pour la réparation, vers un emplacement 'A' pour ramasser une clef à molette (Get-spanner), puis se déplacer à un emplacement 'B' pour prendre une échelle et la transporter au lieu 'C' de la panne (Push-scale). Ensuite, le robot devrait escalader l'échelle (Climb-scale), extraire le boulon défectueux du filtre avec la clef (Extract-bolt) et le remplacer.

La spécification des opérateurs permis dans le 'repairer domain' est présentée dans la figure 28. Comme pour les deux domaines d'études précédents, nous avons pris en compte dix cas d’examles aléatoires pour chaque degré de complexité. Nous exposons les résultats expérimentaux relatifs aux temps d'exécutions dans les tableaux 23-28. Les courbes comparatives entre 'Graphplan' et 'Graphplan-Ordres Partiels' sont illustrés dans la figure 30. À titre d'exemple, la figure 29 Schématise un cas aléatoire de conditions initiales. Le Tableau 22 montre les différents plans d'actions menant aux objectifs - calculés par rapport à la complexité - à partir de ces conditions de départ.

Suite aux essais expérimentaux dans les trois domaines, nous allons évaluer notre combinaison hybride par rapport à Graphplan. Comme les résultats des tests sont tous sur la même longueur d'onde dans les trois domaines, nous avons préféré laisser la discussion générale à la dernière section pour clore ce chapitre.
(operator GOTO
  (params (robot ROBOT) (<x> PLACE) (<y> PLACE))
  (preconds (on-floor <robot>) (at <robot> <y>))
  (effects (del at <robot> <y>) (at <robot> <x>)))

(operator CLIMB-SCALE
  (params (robot ROBOT) (scale SCALE) (<x> PLACE))
  (preconds (at <scale> <x>) (at <robot> <x>))
  (effects (on <scale> <robot> <x>) (del on-floor <robot>)))

(operator PUSH-SCALE
  (params (robot ROBOT) (scale SCALE) (<x> PLACE) (<y> PLACE))
  (preconds (at <scale> <x>) (at <robot> <y>) (on-floor <robot>))
  (effects (at <robot> <x>) (at <scale> <x>) (del at <robot> <y>) (del at <scale> <y>)))

(operator GET-SPANNER
  (params (robot ROBOT) (spanner SPANNER) (<y> PLACE))
  (preconds (at <spanner> <y>) (at <robot> <y>))
  (effects (has <spanner> <robot>) (del at <spanner> <y>)))

(operator EXTRACT-BOLT
  (params (robot ROBOT) (scale SCALE) (spanner SPANNER) (bolt BOLT) (<y> PLACE))
  (preconds (has <spanner> <robot>) (at <bolt> <y>) (on <scale> <robot> <y>))
  (effects (has <bolt> <robot>)))

Figure 28: 'repairer domain' – spécification des opérateurs.

Paramètres : (pack1 OBJECT)
(planet1 AIRPLANE)
(truck1 TRUCK) (truck2 TRUCK)
(paris CITY) (london CITY)
(loc1 LOCATION) (loc2 LOCATION)
(airport1 AIRPORT) (airport1 LOCATION)
(airport2 AIRPORT) (airport2 LOCATION)

Conditions initiales: (at pack1 loc1)
(at truck1 loc1)
(at truck2 loc2)
(at planet1 airport1)
(loc-at loc2 london)
(loc-at loc1 paris)
(loc-at airport1 paris)
(loc-at airport2 london)

Figure 29: 'repairer domain' – un exemple de conditions initiales aléatoires.
<table>
<thead>
<tr>
<th>Complexité</th>
<th>Objectifs</th>
<th>Plan d'actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>at robot1 place2</td>
<td>1 GOTO_robot1_place2_place1</td>
</tr>
<tr>
<td>2</td>
<td>at robot1 place4, has spanner1 robot1</td>
<td>1 GOTO_robot1_place4_place1, 2 GET-SPANNER_robot1_spanner1_place4</td>
</tr>
<tr>
<td>3</td>
<td>has spanner1 robot1, at robot1 place2</td>
<td>1 GOTO_robot1_place4_place1, 2 GET-SPANNER_robot1_spanner1_place4, 3 GOTO_robot1_place2_place4</td>
</tr>
<tr>
<td>4</td>
<td>has spanner1 robot1, at robot1 place3, at scale1 place3</td>
<td>1 GOTO_robot1_place2_place1, 2 PUSH-SCALE_robot1_scale1_place4_place2, 3 GET-SPANNER_robot1_spanner1_place2, 4 PUSH-SCALE_robot1_scale1_place3_place2</td>
</tr>
<tr>
<td>5</td>
<td>has spanner1 robot1, at scale1 place3, on scale1 robot1 place3</td>
<td>1 GOTO_robot1_place4_place1, 2 GET-SPANNER_robot1_spanner1_place4, 3 GOTO_robot1_place2_place4, 4 PUSH-SCALE_robot1_scale1_place3_place2, 5 CLIMB-SCALE_robot1_scale1_place3</td>
</tr>
<tr>
<td>6</td>
<td>has bolt1 robot1</td>
<td>1 GOTO_robot1_place4_place1, 2 GET-SPANNER_robot1_spanner1_place4, 3 GOTO_robot1_place2_place4, 4 PUSH-SCALE_robot1_scale1_place3_place2, 5 CLIMB-SCALE_robot1_scale1_place3, 6 EXTRACT-BOLT_robot1_scale1_spanner1_bolt1_place3</td>
</tr>
</tbody>
</table>

*Tableau 22: 'repairer domain' - exemple de buts reliés à la complexité.*
**Tableau 23:** 'repaire domain' – les temps d'exécution pour une complexité C1.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,08</td>
<td>0,06</td>
<td>0,07</td>
<td>0,08</td>
<td>0,09</td>
<td>0,06</td>
<td>0,07</td>
<td>0,08</td>
<td>0,09</td>
<td>0,09</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Tableau 24:** 'repaire domain' – les temps d'exécution pour une complexité C2.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,17</td>
<td>0,012</td>
<td>0,14</td>
<td>0,13</td>
<td>0,17</td>
<td>0,16</td>
<td>0,17</td>
<td>0,18</td>
<td>0,17</td>
<td>0,19</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Tableau 25:** 'repaire domain' – les temps d'exécution pour une complexité C3.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,31</td>
<td>0,45</td>
<td>0,37</td>
<td>0,29</td>
<td>0,34</td>
<td>0,41</td>
<td>0,32</td>
<td>0,38</td>
<td>0,43</td>
<td>0,47</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Tableau 26: ‘repaier domain’ – les temps d’exécution pour une complexité C4.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>0,67</td>
<td>0,89</td>
<td>0,77</td>
<td>0,82</td>
<td>0,91</td>
<td>0,94</td>
<td>0,88</td>
<td>0,84</td>
<td>0,79</td>
<td>0,82</td>
</tr>
<tr>
<td>Moyenne</td>
<td>0,82</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Tableau 27: ‘repaier domain’ – les temps d’exécution pour une complexité C5.

<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>1,32</td>
<td>1,17</td>
<td>1,42</td>
<td>1,38</td>
<td>1,15</td>
<td>1,29</td>
<td>1,31</td>
<td>1,4</td>
<td>1,46</td>
<td>1,37</td>
</tr>
<tr>
<td>Moyenne</td>
<td>1,33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Exemples</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
<th>E5</th>
<th>E6</th>
<th>E7</th>
<th>E8</th>
<th>E9</th>
<th>E10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temps (s)</td>
<td>1,83</td>
<td>1,75</td>
<td>1,92</td>
<td>1,79</td>
<td>1,88</td>
<td>1,94</td>
<td>2,01</td>
<td>1,97</td>
<td>1,79</td>
<td>1,86</td>
</tr>
<tr>
<td>Moyenne</td>
<td>1,87</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tableau 29: 'repairer domain' – comparaison des temps d'exécution par rapport à la complexité.

<table>
<thead>
<tr>
<th>Complexité</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphplan</td>
<td>0,01</td>
<td>0,02</td>
<td>0,03</td>
<td>0,04</td>
<td>0,06</td>
<td>0,07</td>
</tr>
<tr>
<td>Graphplan-OrdresPartiel</td>
<td>0,08</td>
<td>0,15</td>
<td>0,38</td>
<td>0,82</td>
<td>1,33</td>
<td>1,87</td>
</tr>
</tbody>
</table>

Figure 30: 'repairer domain' – courbes des temps d'exécution par rapport à la complexité.
4.4 Discussion

Nous avons procédé à une évaluation expérimentale des deux planificateurs sur des cas aléatoires de problèmes dans les trois domaines décrits précédemment. D’après les résultats des tests, nous pouvons conclure que Graphplan demeure plus rapide, donc plus efficace – en temps d’exécution – que notre combinaison hybride. Nous avons remarqué que sur les cas de problèmes à faible complexité (entre un et trois) ‘Graphplan-Ordres_Partiel’ est généralement quatre à six fois plus long que Graphplan qui prend le large au fur et à mesure que la complexité augmente pour être dix à quinze fois plus rapide pour un degré de complexité supérieur ou égale à six (c’est-à-dire, le plan minimal contient six actions).

Il est vrai que nous avons réussi à appliquer une bonne technique de réduction de l’espace d’états que nous avons parfaitement greffé au sein de Graphplan métamorphosant ainsi sa philosophie de fonctionnement, mais, en absence de tout procédé basé sur le raisonnement – quand il s’agit de choisir l’action à exécuter – notre exploration en profondeur se produit quelque peu à l’aveuglette. Ce qui nous fait perdre trop de temps, car nous explorons dans certains cas beaucoup de séquences inutiles.
CONCLUSION

Rappelons d’abord que l’objectif de ce mémoire est d’évaluer une nouvelle idée de planification basée sur une approche ‘hybride’. Cette approche est le résultat d’un amalgame entre Graphplan et la théorie des ordres partiels appliquée dans la vérification des modèles de systèmes concurrents.


Nous avons implémenté notre approche hybride en langage C et nous l’avons testée dans trois domaines d’étude différents. Les essais comparatifs entre Graphplan et notre planificateur ont montré que Graphplan demeure plus efficace quant au temps d’exécution. Il est vrai que notre nouvelle méthode est nettement meilleure qu’une simple recherche en profondeur à chaînage avant, car nous sommes parvenu à régler le problème de l’explosion combinatoire grâce aux ensembles persistants combinés avec les ensembles ‘sleep’. Donc, nous avons réussi à réduire considérablement – par rapport à la recherche classique – le temps d’exécution ainsi que l’espace mémoire utilisé pour le stockage des données. Cependant, étant donné que notre planificateur n’utilise aucune heuristique et n’adopte aucune stratégie évoluée pour contrôler
la recherche de plans, celle-ci – bien qu'elle se fasse dans un espace d'états réduit – reste encore quelque peu anarchique. Ceci s'explique par le fait que les actions retenues pour être appliquées (suite à la sélection faite au moyen des ordres partiels) sont exécutées selon un ordre aléatoire. Nous ne disposons pas d'un mécanisme de raisonnement capable de choisir une action particulière et de la favoriser – selon certains critères – par rapport aux autres actions retenues. C'est pour cette raison que notre planificateur gaspille trop de temps à explorer beaucoup de séquences peu intéressantes, c'est-à-dire qui ne le rapprochent pas des buts recherchés.

En dépit du résultat peu favorable de notre travail, nous avons pu constater le rôle capital des mécanismes évolués de contrôle de la recherche qui se basent sur le raisonnement et sur la connaissance acquise du domaine. Nous pouvons envisager la possibilité d'utiliser des stratégies de contrôle comme les formules logiques d'optimisation de la recherche, par exemple, ou encore, des heuristiques faiblement couplées aux domaines d'étude qui ne sont pas spécifiques ou uniquement efficaces pour un type particulier de domaines.
ANNEXE

1. Instructions d’installation

1.2 Installation sous un environnement UNIX

a. Copiez tout le répertoire G_PO contenu dans la disquette annexée.

b. À partir d’un terminal et en étant sous le répertoire G_PO, lancez la commande ‘make’ :
   \G_PO\make

c. Si vous décidez de restructurer le contenu du répertoire G_PO en créant des sous répertoires dedans, assurez vous que tous les fichiers munis des extensions ‘_fact’ et ‘_ops’ se trouvent dans le même sous répertoire que l’exécutable ‘Graphplan_PO’ généré après la commande ‘make’.

d. L’application est ainsi prête pour l’utilisation.

1.3 Installation sous un environnement WINDOWS

Comme l’application a été conçue avec un compilateur GNU fonctionnant sous UNIX, l’installation sous un environnement WINDOWS nécessite un passage par un développeur C++ opérant sous WINDOWS.

a. À partir d’un développeur C++ (comme Microsoft Visual C++ ou Visual Studio C++, par exemple) copiez tout le contenu du répertoire G_PO dans un nouveau projet que vous devez créer préalablement.

b. Lancer une compilation globale afin de générer les fichiers objet et ceux exécutables.
c. Assurez vous que tous les fichiers munis des extensions ‘_fact’ et ‘_ops’ se trouvent dans
le même répertoire que l’exécutable ‘Graphplan_PO’ généré après la compilation globale.

d. Lancer l’exécution de l’application à partir du fichier ‘graphplan.c’.

2. Instructions d’utilisation

2.1 Utilisation du planificateur sur des problèmes déjà définis

Si vous désirez lancer le planificateur sur des cas de problèmes dont les domaines sont
définis (rocket, repairer, logistic, monkey, mblocks), les fichiers ‘_ops’ contiennent déjà les
spécifications des opérateurs et les fichier ‘_fact’ comportent les définitions des contraintes
liées à ce cas de problème (paramètres, conditions initiales et objectifs). Voici les étapes :

a. S’assurer que les fichiers munis des extensions ‘_fact’ et ‘_ops’ – et relatifs au problème – se trouvent dans le même répertoire que l’exécutable ‘Graphplan_PO’.

b. Si vous souhaitez changer les conditions initiales du problème ou les buts à atteindre, il faudrait le faire dans le fichier ‘_fact’ relatif au problème (par exemple, ‘roc_fact’ pour le ‘rocket domain’). N’oubliez pas de sauvegarder les modifications suite à n’importe quel changement du contenu du fichier.

c. Lancez l’exécution avec la commande :

   `\G_PO\Graphplan_PO -o <nom du fichier>_ops -f <nom du fichier>_fact`

d. À titre d’exemple, la figure 31 montre un cas d’exécution pour un problème dans le ‘rocket
domain’. Après avoir lancé la commande d’exécution, le planificateur demande à
Figure 31: Un exemple d'exécution du planificateur sur un cas de problème dans le 'rocket domain'.
l'utilisateur de spécifier le mode de planification qui peut être (1) le mode 'classic' où c'est la version classique de Graphplan qui s'exécute ; ou bien (2) le mode 'partial order' où c'est la version hybride 'Graphplan-ordres_partiels' qui s'exécute.

2.2 Pour définir de nouveaux problèmes

a. Spécifier les opérateurs en format UCPOP dans un fichier muni d'une extension '_ops'. Vous pouvez vous inspirer des modèles d'exemples existants (roc_ops, logistics_ops, monkey_ops, mblocks_ops, repairer_ops).

b. Définir les paramètres (objets), les conditions initiales et les objectifs à satisfaire dans un fichier muni d'une extension '_fact'. Vous pouvez également vous inspirer des modèles d'exemples existants (roc_fact, logistics_fact, monkey_fact, mblocks_fact, repairer_fact).

c. S'assurer que les deux fichiers munis des extensions '_fact' et '_ops' se trouvent dans le même répertoire que l'exécutable 'Graphplan_PO'.

d. Lancez l'exécution avec la commande :

```
\G_PO\Graphplan_PO -o <nom du fichier>_ops -f <nom du fichier>_fact
```

3. Description du code source

Voici la description des principaux fichiers que contient la disquette annexée à ce mémoire :

- **graphplan.c** : contient les procédures de démarrage et de la création du graphe. Ce fichier a été écrit par F Blum et B. Furst [BF97]. Nous avons modifié uniquement le code relié à l'interface.

- **project.c** : contient les procédures de la génération de l'espace d'états (création des états, exécutions des actions, etc.)
**persist.c** : contient les procédures relatives à la construction des ensembles persistants et les ensembles 'sleep'.

**planner.c** : contient les procédures de la planification classique de Graphplan. Ce fichier a été écrit par F Blum et B. Furst [BF97].

**utilities.c** : contient les procédures utilitaires associées au processus de la planification classique de Graphplan. Ce fichier a été écrit par F Blum et B. Furst [BF97].

**Les fichiers \'_fact\'** : comme roc_fact, logistics_fact, monkey_fact, mblocks_fact et repairer_fact. Ce sont des exemples de cas de problèmes dans divers domaines sur les quels le planificateur peut opérer.

**Les fichiers \'_ops\'** : ils contiennent les spécifications des opérateurs.

**Les fichiers \'test_*\'.cpp\'** : ce sont les programmes de la génération aléatoire des conditions initiales et des objectifs (comme 'test_rocket.cpp, par exemple).


