Fondation d'un planificateur robotique intégrant le symbolique et le géométrique

Fabien Gravot

Thèse

préparée au

Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

en vue de l'obtention du

Doctorat de l'Université Paul Sabatier de Toulouse

Spécialité : Informatique

par

Fabien GRAVOT

---

aSyMov : Fondation d'un planificateur robotique intégrant le symbolique et le géométrique.

Soutenue le 24 Mars 2004, devant le jury :

Président :	**Raja**	**Chatila**	Directeur de Recherche CNRS
Directeur de thèse :	**Rachid**	**ALAMI**	Directeur de Recherche CNRS
Rapporteurs :	**Lydia**	**Kavraki**	Professeur à l'Université Rice, USA
	Emmanuel	**Mazer**	Directeur de Recherche CNRS, IMAG
Examinateurs :	**Joachim**	**Hertzberg**	Docteur à l'Institut Fraunhofer, AIS, Professeur à l'Université de Bonn, RFA
	Michel	**Courdesses**	Professeur à l'Université Paul Sabatier

LAAS-CNRS
7, Avenue du Colonel Roche
31077 Toulouse Cedex 4

Avant Propos

Le travail présenté dans ce mémoire à été conduit au sein du groupe de « Robotique et Intelligence Artificielle » du LAAS-CNRS à Toulouse. Je tiens à remercier l'accueil que m'a prodigué Malik Ghallab directeur du laboratoire qui m'a donné de précieux conseils et remarques tout au long de ma thèse. Je remercie aussi Raja Chatila responsable du pôle Robots et Systèmes Autonomes pour son accueil et ses encouragements.

Je voudrais exprimer ma gratitude envers mon directeur de thèse, Rachid Alami, dont j'ai pu apprécier la compétence, l'enthousiasme, la confiance qu'il m'a donnée et son soutien inestimable lors des moments clés de cette thèse. Je m'excuse aussi auprès de sa famille pour les soirées que je leur ai volées.

J'adresse aussi ma reconnaissance à Emmanuel Mazer et à Lydia Kavraki qui ont accepté d'être les rapporteurs de cette thèse soutenue devant le jury auquel ont aussi participé Raja Chatila, son président, Joachim Hertzberg, Michel Courdesses et Rachid Alami. Qu'ils soient tous assurés de ma gratitude pour avoir jugé et enrichi mon travail.

Le travail effectué pendant ces trois ans, n'est pas un travail isolé, mais représente le fruit d'une collaboration scientifique et humaine riche et fructueuse. Je tiens à remercier plus particulièrement Stéphane Cambon qui a grandement participé à l'élaboration de ces travaux et qui va continuer à les enrichir. Mais je dois aussi remercier ceux qui m'ont précédé et inspiré : Juan Cortès, Anis Sahbani, Julien Pettre, Sílvia Botelho, Romain Trinquart et ceux qui me suivent Solange Lemai, Léonard Jaillet avec qui j'ai eu de fructueux échanges. Le travail de thèse en général solitaire, a été pour moi un moment d'échange d'idées et de collaboration que je ne risque pas d'oublier.

Enfin je tiens à remercier tous ceux qui ont aussi contribué à rendre ces années chaleureuses et agréables et plus particulièrement Jérémi Gancet, Yannick Larvor, Emmanuel Guéré, Williams Paquier, Patrick Danes et Emmanuel Zenou. Je pourrais continuer encore longtemps à citer tous les gens qui m'ont fait apprécier la vie dans la ville rose, mais il me serait alors impossible de présenter mes travaux avec le peu de pages qu'il me resterait. Je vais donc finir en remerciant particulièrement Nicola Siméon qui m'a consacré son temps, sa patience, son savoir et son énergie pour faire progresser mes travaux.

Sommaire

Chapitre 1

Introduction

Cette thèse a été préparée au sein du groupe de « Robotique et d'Intelligence Artificielle » du LAAS-CNRS. La robotique a pour vocation le développement de robots autonomes. Par autonome, on sous-entend qu'un robot est capable de percevoir son environnement, de raisonner pour accomplir une mission, et d'agir sur ce même environnement. Cette thèse porte particulièrement sur le problème de l'intelligence et du raisonnement de ces robots et plus particulièrement sur les problèmes de planification.

En robotique le terme « planification » désigne deux domaines de recherche bien distincts par leurs approches et les domaines traités : la planification de mouvements et la planification de tâches.

La planification de mouvement permet de connaître l'ensemble des commandes de déplacement d'un robot permettant d'aller d'un endroit à un autre tout en évitant les obstacles. Les techniques de la planification de mouvement ont connu des avancées importantes ces dernières années. On notera par exemple, la recherche de trajectoire sans collision, la manipulation coordonnée d'objets, les techniques de saisie, et les techniques de planification de tâches de manipulation où l'environnement contraint la recherche de positions intermédiaires de pose de l'objet manipulé.

La planification de tâches, quant à elle, s'intéresse à des aspects plus abstraits. En conséquence les actions ne sont pas forcément simples à appliquer. En effet l'action de saisie d'un objet quelconque peut nécessiter un ensemble de mouvements complexes dépendant de l'environnement.

C'est ainsi, que pour accomplir un ensemble d'actions de haut niveau, il est nécessaire de coupler ces deux approches.

Une approche *« naïve »* consiste à supposer qu'il est possible de traiter ces problèmes de manière hiérarchique, la planification de mouvements n'étant utilisée que pour résoudre les tâches élémentaires trouvées par la planification de tâches. Cette séparation des domaines n'est pas faisable pour le cas

d'un ou plusieurs robots qui manipulent des objets dans un environnement contraint.

Dans le cadre de cette thèse nous avons développé un planificateur nommé « aSyMov » (a Symbolic Move3D[1]) qui a pour vocation d'offrir un cadre innovant à cette interaction entre ces deux techniques de planification. L'interaction sera si forte que les informations tant symboliques que géométriques seront prises en compte à chaque étape de ce nouveau planificateur.

Ainsi la topologie de l'environnement explorée par des techniques de la planification de mouvements et les informations symboliques explorées par des techniques de la planification de tâches vont guider le choix des actions.

De même l'exploration de la topologie de l'environnement sera influencée par ces deux aspects afin d'orienter cette recherche principalement sur les données nécessaires à la résolution d'une mission donnée.

Cette interaction va permettre d'aborder des problèmes encore non résolus en robotique dans le cas de plusieurs robots. « aSyMov » pourra prendre en compte simultanément : le choix des positions intermédiaires, la manipulation avec définition de positions de prise et de pose continues ainsi que des informations symboliques n'ayant pas de rapport avec la géométrie des robots.

Le manuscrit va dans un premier temps faire un état de l'art des techniques utilisées pour la planification de mouvements. Certaines sont utilisées dans l'implémentation actuelle d'aSyMov. Mais cet état de l'art ne donne pas un moyen de combiner ces techniques et ne définit pas un espace de recherche permettant d'exprimer toutes les actions que nous voulons envisager : mouvements, manipulations, assemblages avec plusieurs robots et plusieurs objets.

Une partie des contributions de cette thèse porte sur ces deux points : la représentation de l'espace de recherche par l'intermédiaire de *« Graphes des Cinématiques Élémentaires »* (GCEs) et la combinaison de techniques de planifications avec l'utilisation d'un système de *« règles heuristiques »*.

Le chapitre 3 présentera notre nouvel espace de recherche : les *« Graphes des Cinématiques Élémentaires »* (GCEs). Ceux-ci vont nous permettre de définir l'ensemble des actions de déplacement (à travers des roadmaps) et des actions de manipulation ou d'assemblage (liens entre roadmaps).

Les GCEs sont basés sur l'utilisation de plusieurs roadmaps qui résolvent de manière indépendante les différents mouvements possibles. Cette séparation permet de réduire la complexité de la recherche d'une solution globale.

Le chapitre suivant nous permettra de définir un moyen de guider l'apprentissage de la topologie à travers les différentes roadmaps utilisées. Pour cela nous utiliserons un système de *« règles heuristiques »* qui choisira *« où »* et *« comment »* apprendre.

Ce chapitre présentera aussi quelques innovations permettant de traiter dans un cadre général les différentes techniques de planification de mouvement. Nous avons notamment introduit l'utilisation

[1] *« Move3D »* est le planificateur de mouvements développé au sein du laboratoire.

des *« roadmaps heuristiques »* et des *« roadmaps relatives »*. Les roadmaps heuristiques permettent de réduire l'espace de recherche, de trouver *« où »* il faut chercher. Les *« roadmaps relatives »* permettent d'étudier et de réutiliser les mouvements contraints tel que la saisie d'objets.

Après avoir décrit les méthodes de la planification de mouvements, nous verrons dans le chapitre 5 les techniques de la planification de tâches.

Ce chapitre nous permettra aussi de définir une manière originale de lier le domaine géométrique au domaine symbolique. Cette liaison sera faite à travers un certain nombre de types et de prédicats qui vont permettre de définir des *« positions symboliques »*. Une *« position symbolique »* représentera une fonction associée à un sous-ensemble de l'espace des configurations.

Le chapitre 6 va nous permettre de présenter enfin l'architecture et les algorithmes de notre planificateur. Il constitue notre contribution principale.

ASyMov effectue une recherche *en avant* dans l'espace d'état. L'état d'aSyMov a la particularité de représenter l'ensemble des instanciations géométriques possibles pour un état symbolique donné. Ces instances seront dépendantes des actions qui ont mené à l'état courant.

Nous avons établi une procédure de validation des actions qui va essayer de minimiser les instances géométriques à vérifier. Ce ne sera que quand les actions vont contraindre la géométrie qu'une propagation arrière de ces contraintes sera faite pour trouver de nouvelles instances valides.

Nous avons aussi adapté les algorithmes de guidage de la recherche pour prendre en compte les informations symboliques et géométriques. Ainsi l'heuristique correspond à une longueur de plan symbolique augmentée par les contraintes topologiques.

Enfin nous décrivons comment il est possible de combiner à la fois l'apprentissage de la topologie de l'environnement et la recherche d'une solution avec le niveau actuel des connaissances déjà acquises.

Le chapitre 7 présentera plusieurs problèmes qui ont été résolus par aSyMov. Nous ferons remarquer à ce moment l'importance qu'il y a à prendre en compte les contraintes symboliques et géométriques à tous les niveaux de la planification, nous soulignerons aussi les limites de notre système et les méthodes utilisables pour les dépasser.

Chapitre 2

Planification géométrique pour la Robotique.

La planification de mouvements permet de prendre en compte les contraintes géométriques pour trouver un chemin sans collision pour un robot dans un environnement quelconque. Elle a fortement progressé ces dernières années.

Une des tâches élémentaires qu'un robot doit être capable de faire, c'est d'interagir avec son environnement notamment par la manipulation. Cette tâche est intrinsèquement liée aux aspects géométriques du problème. De récents travaux en manipulation [Sahbani 03] ont montré la difficulté qu'un planificateur devait surmonter pour trouver des positions intermédiaires intéressantes quand les possibilités de placement et de prises sont infinies.

Ainsi par le terme de *« planification géométrique »*, je vais désigner l'ensemble des techniques qui prennent en compte de manière explicite la géométrie des robots et de leur environnement ainsi que leurs contraintes cinématiques de mouvement.

La capacité de se mouvoir étant une caractéristique essentielle d'un système robotique, indispensable pour effectuer des tâches de manipulation, nous commencerons par l'étudier dans un premier temps, ainsi que plusieurs variantes permettant d'étudier des problèmes de plus en plus complexes.

Puis nous terminerons sur un état de l'art des approches géométriques de la planification de tâches de manipulation.

2.1 Planification de mouvement

Un problème de planification de mouvement consiste à calculer une trajectoire pour un robot entre une position initiale et une position finale. Il peut être sommairement posé de la façon suivante :

Etant donné un robot évoluant dans un environnement parsemé d'obstacles, trouver s'il existe un mouvement ou une succession de mouvements amenant ce robot d'une position à une autre, tout en respectant ses contraintes cinématiques et en évitant les obstacles.

L'algorithmique de la planification de mouvement débute avec l'introduction de la notion d'espace de configurations [Lozano-Pérez 83]. Lozano-Pérez a substitué la recherche d'un chemin sans collision pour un robot évoluant parmi des obstacles par la recherche de chemin d'un point évoluant dans l'espace de *« configurations »* du robot.

Définition 1 *La* « configuration » *d'un robot représente l'état géométrique du robot. Cela correspond communément à une instance de ses degrés de libertés.*

Les degrés de libertés ne correspondent pas forcément à des articulations. Pour représenter certaines cinématiques il faut introduire des données de vitesses ou d'accélérations [Lamiraux 01].

De plus ces degrés de libertés ne sont pas forcément indépendants les uns des autres comme nous le verrons pour l'étude des chaînes cinématiques fermées (§ 2.5).

Depuis la définition de cet espace de recherche, la planification de mouvement est devenue rapidement un domaine de recherche très actif, se focalisant sur quatre approches principales :

- la décomposition cellulaire [Schwartz 83];
- les méthodes de rétraction [O'Dùnlaing 82];
- les champs de potentiels [Khatib 86];
- la construction de graphe (*roadmaps*) [Nilsson 69].

Dans sa version la plus simple, le problème est intrinsèquement difficile [Reif 79], sa complexité croit exponentiellement avec le nombre de degrés de liberté du robot [Canny 87].

Traditionnellement l'algorithmique du mouvement conduisait soit à des méthodes exactes et complètes, soit à des méthodes heuristiques. La complexité des premières rend leur utilisation rédhibitoire dès qu'il s'agit de systèmes de dimension élevée. Les secondes souffrent des inconvénients liés à la non complétude.

Plus récemment, plusieurs approches non déterministes sont apparues, elles ne reposent plus sur une construction préalable d'une représentation exacte ou approchée de l'espace de configurations. Certaines de ces méthodes se basent sur l'exploration de l'espace de configurations admissibles d'autres visent à capturer sa connectivité. Ces nouvelles méthodes de planification dites *« méthodes probabilistes »*, permettent de faire face à la complexité exponentielle du problème.

Ces méthodes vérifient une propriété de complétude probabiliste (garantissant en probabilité de trouver une solution en temps fini, lorsqu'elle existe). Certes cette propriété est moins forte que la complétude déterministe vérifiée par les méthodes exactes mais ces nouvelles approches sont désormais moins sensibles à la dimension de l'espace de recherche et s'avèrent très efficaces en pratique. Nous allons étudier particulièrement cette approche qui va être utilisée tout au long de ce document.

Dans les sections suivantes, nous présenterons ces approches ainsi que quelques variantes du schéma classique proposées dans la littérature. Pour plus de détails sur l'ensemble des contributions, notamment sur les autres méthodes, nous renvoyons les lecteurs vers le livre de référence dans ce domaine [Latombe 91] ainsi qu'à deux autres ouvrages plus récents [Laumond 98] et [Gupta 98].

2.2 Les méthodes probabilistes

Ces méthodes ont été introduites simultanément par Kavraki et Latombe sous le nom de *PRM* (Probabilistic Roadmap Methodes) [Kavraki 95, Kavraki 96] et par *Š*vestka et Overmars sous le nom de de *PPP* (Probabilistic Path Planner) [Švestka 96]. Depuis, plusieurs variantes de ces méthodes ont été proposées dans la littérature mais toutes se basent sur le même concept.

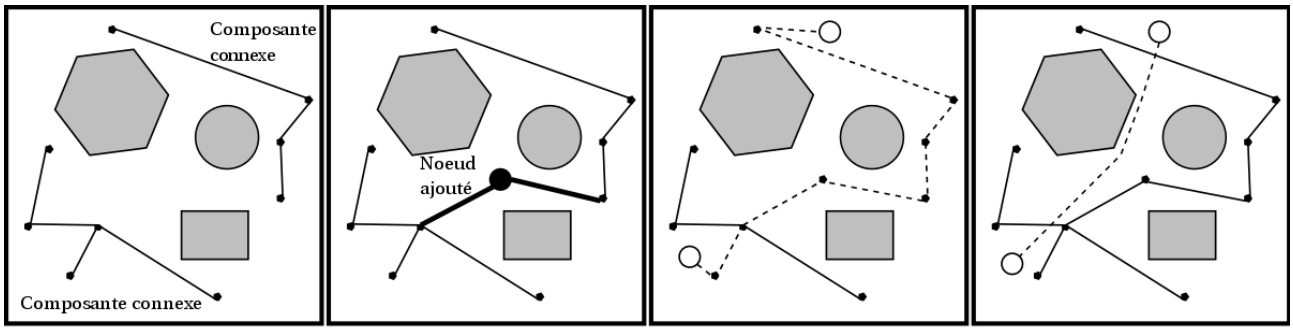


Figure 2.1: Les trois étapes des *PRM*

La technique développée se décompose en trois phases (figure 2.1) : la phase *« d'apprentissage »* de la topologie et les phases de recherche et de lissage d'une solution.

2.2.1 La phase « d'apprentissage ».

Le principe sous-jacent aux méthodes probabiliste est simple. Il consiste à capturer la connectivité de l'espace libre par un graphe (ou un arbre) appelé *« roadmap »*, construit à partir de points choisis aléatoirement dans l'espace des configurations.

Définition 2 *Une* « méthode locale » *permet de tester l'existence d'un chemin faisable entre deux points donnés du graphe. Elle définit un mouvement d'un robot entre ces deux configurations, i.e. un changement continue dans l'espace des configurations. La méthode locale utilisée prend en considération les différentes contraintes mécaniques et cinématiques du système.*

Une « méthode locale » *est dite non orientée si l'ensemble des configurations qu'elle parcourt d'un point A à un point B est le même que celui permettant d'aller du point B au point A.*

Définition 3 *Une* « roadmap » *est un graphe orienté ou non suivant la nature de la* « méthode locale »*, dont les nœuds correspondent à des configurations choisies aléatoirement et les arcs correspondent aux* « méthodes locales »*.*

Un *« détecteur de collision »* permet de vérifier si les configurations des arcs sont admissibles (sans collision). Ainsi deux nœuds d'une *roadmap* sont reliés par un arc lorsque le chemin calculé est sans collision et respecte les contraintes cinématiques du robot.

Définition 4 *Une* « composante connexe » *est un sous-ensemble de nœuds d'une* « roadmap » *qui sont connectés directement ou indirectement les uns aux autres (figure 2.1.a).*

La « *roadmap* » capturant la topologie de l'espace libre peut être construite dans une étape préalable afin de résoudre efficacement plusieurs requêtes de planification de mouvement. Elle peut également être enrichie au fur et à mesure que le planificateur résout un problème.

Plusieurs points sont à détailler dans ce schéma global : Comment choisir aléatoirement une configuration ? Quelle méthode locale utiliser ? Comment choisir les nœuds à connecter ? etc.

La méthode de tirage aléatoire des configurations peut être un point clef de la résolution d'un problème contraint. Un environnement peut être constitué de grandes pièces et de passages étroits, dans ce cas il est important d'avoir des nœuds pour franchir ces passages étroits même si ceux-ci ne représentent qu'une faible part de l'espace de configuration. Il faut alors « forcer un peu sa chance » (§ 2.3.1).

La méthode locale est imposée en général par les contraintes cinématiques du robot. Néanmoins, son choix peut avoir une influence importante sur les performances de l'algorithme [Laumond 00]. Elle est appelée un très grand nombre de fois lors de la construction de la *roadmap*. Elle doit donc être rapide et efficace.

Le choix des nœuds à connecter au nouveau nœud généré aléatoirement est également un paramètre important dans cette phase d'apprentissage. En effet, ce choix influencera la structure de la *roadmap* ainsi que les performances de l'algorithme. Ce choix dicte également le nombre d'appels à la méthode locale (qui peut être coûteuse en terme de temps de calcul). Généralement, ce choix est fait en fonction d'une distance entre les nœuds. La fonction distance utilisée peut devenir un paramètre critique difficile à établir dans l'espace des configurations.

Dans un environnement statique, l'interdiction des cycles est souvent envisagée afin de minimiser le nombre d'appels à la méthode locale. Cette contrainte doit être relâchée quand nous voulons garder une information de connexité, même quand certains arcs sont invalidés par des obstacles mobiles.

2.2.2 La phase de recherche

Une *roadmap* qui reflète la connexité de l'espace libre, peut être utilisée pour la résolution des différentes requêtes de planification. La phase de recherche consiste à relier dans un premier temps la configuration de départ et le but aux nœuds de la *roadmap* en utilisant la méthode locale. En cas de succès, une recherche de chemin dans la *roadmap* est effectuée. Ce chemin, quand il existe, est une séquence de chemins locaux. Un algorithme de type A^* peut être utilisé pour la recherche du chemin dans la *roadmap*.

Si les deux configurations, départ et but, n'appartiennent pas à la même *composante connexe*, la phase « *d'apprentissage* » de la *roadmap* peut être effectuée [Kavraki 95]. Cette phase est arrêtée une fois que les deux configurations sont dans la même composante connexe, ou si un critère d'arrêt est vérifié. Cette décision d'arrêter cette phase peut être donnée par un nombre maximal de nœuds à rajouter à la *roadmap*.

Dans le cas d'un environnement dynamique, cette phase doit aussi vérifier la validité des chemins trouvés par rapport aux paramètres non pris en compte lors de la phase d'apprentissage.

2.2.3 Optimisation et lissage des chemins

En raison du caractère probabiliste du planificateur, le chemin solution est généralement loin d'être optimal. Il peut être long et irrégulier. Une phase d'optimisation locale ou plutôt de lissage s'avère donc nécessaire.

Une première méthode a été introduite dans [Laumond 94]. Elle est plutôt simple et permet de faire des optimisations de grande envergure.

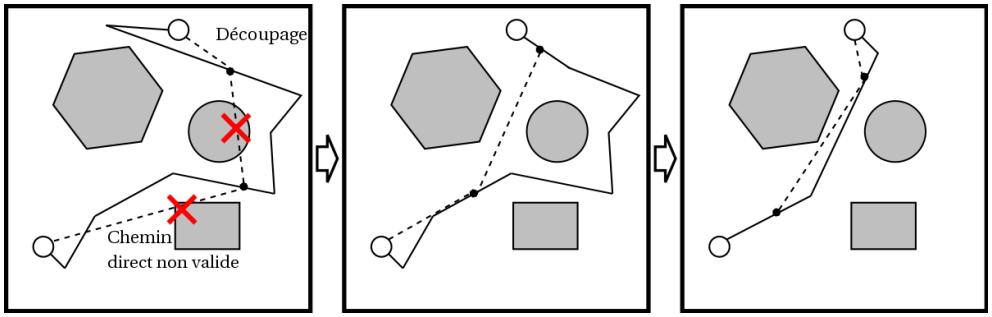


Figure 2.2: Optimisation de trajectoire par découpage aléatoire en trois parties.

Elle consiste à découper aléatoirement le chemin à lisser en trois parties, et à essayer de remplacer chacune d'elles par le chemin local reliant leurs extrémités. Dans le cas où l'une des portions calculées est plus courte que la séquence initiale des chemins locaux, elle remplace la séquence initiale (figure 2.2).

Une autre méthode possible, plus propice aux optimisations locales, utilise deux segments consécutifs $\widetilde{AB}$ et $\widetilde{BC}$ de la trajectoire (figure 2.3) et essaie de les rapprocher de $\widetilde{AC}$.

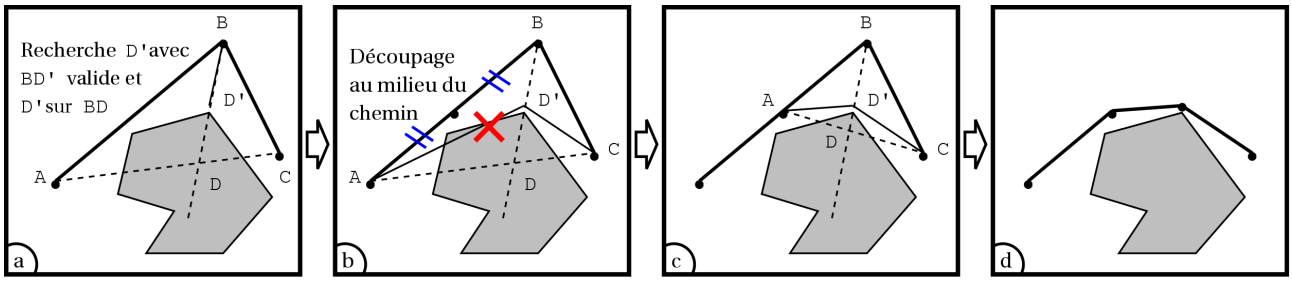


Figure 2.3: Deux phases d'optimisation locale. (a) et (b) montre un cas avec collision donc avec création d'un point intermédiaire. (c) et (d) montre un cas sans collision donc un lissage.

Si $\widetilde{AC}$ n'est pas valide alors elle crée un point D tel que :

$$\|\widetilde{BC}\| \times \|\widetilde{AD}\| = \|\widetilde{AB}\| \times \|\widetilde{DC}\| \ (D \in \widetilde{AC}).$$

Elle va alors tenter de rejoindre le point D pour arriver au point D' sans collision.

Si $\widetilde{AD'}$ et $\widetilde{D'C}$ sont valides et plus courts que $\widetilde{AB}$ et $\widetilde{AC}$, ils remplacent alors $\widetilde{AB}$ et $\widetilde{BC}$.

Sinon si $\widetilde{AD'}$ (respectivement $\widetilde{D'C}$) est en collision, le chemin $\widetilde{AB}$ (respectivement $\widetilde{BC}$) est coupé en deux en son milieu (pourvu que les chemins résultant de cette coupure ait au moins une somme de longueur inférieur ou égale au chemin initial). Lors de la prochaine étape d'optimisation ces nouveaux points vont permettre de mieux s'adapter aux formes des obstacle et de s'en rapprocher.

Ce second algorithme bien que générique fonctionne d'autant mieux si la méthode locale utilisée à une propriété de distance et par conséquent que $\|\widetilde{AC}\| \leq \|\widetilde{AB}\| + \|\widetilde{BC}\|$. Son fonctionnement est optimal pour les robots holonome.

Cette opération est répétée jusqu'à ce que le gain de lissage devienne négligeable ou que la durée fixée à la phase d'optimisation soit écoulée.

D'autres variantes existent notamment basées sur une bande élastique où des forces internes imposent une tension à la trajectoire pour la raccourcir comme avec un élastique tandis que des forces externes permettent de repousser la trajectoire en dehors des obstacles [Brock 97].

Ces méthodes sont toutefois assez complexes quand les contraintes cinématiques se superposent aux forces internes comme le montrent de récents travaux [F. Lamiraux 02]. Elles ne sont donc pas toujours généralisables.

2.3 Le tirage aléatoire des nœuds.

De nombreuses variantes et extensions de ce schéma général ont par la suite été proposées afin de capturer plus efficacement la connexité de l'espace libre.

Certains auteurs ont mis en avant l'inconvénient de la stratégie d'échantillonnage uniforme de la méthode classique et ont présenté d'autres stratégies permettant une résolution plus efficace de problèmes difficiles, en particulier lorsqu'il s'agit de capturer la connectivité de régions libres de l'espace des configurations reliées par un passage étroit.

D'autres ont travaillé sur la minimisation de la taille de *roadmaps* afin d'accélérer certaines phases ou de rajouter d'autres paramètres de contrôle de l'algorithme générique.

2.3.1 Les passages étroits.

Les méthodes probabilistes classiques (*PRM* et *PPP*) échantillonnent de façon uniforme l'espace libre. Ce type d'échantillonnage produit une couverture admissible de l'espace libre si les différents obstacles sont répartis de façon uniforme dans l'environnement. Malheureusement, ceci n'est pas toujours le cas. Dans certains cas, l'espace des configurations admissibles se compose d'espaces dégagés connectés par des passages étroits. Générer des nœuds dans ce type de passage est coûteux en terme de temps de calcul, ce qui se traduit aussi par une *roadmap* de taille grande (un grand nombre de nœuds générés). De plus, la probabilité de générer une configuration dans un passage étroit, dans un temps petit (raisonnable), est quasiment nulle. De nombreux auteurs ont suggéré différentes méthodes permettant d'obtenir efficacement une couverture admissible des zones dites *« difficiles »*.

Une première approche [Kavraki 96], concentre l'échantillonnage aléatoire autour des composantes isolées de la *roadmap* afin de capturer plus efficacement les passages étroits. L'idée principale de cet algorithme est de conserver le nombre d'échecs du planificateur local lors de la connexion entre certains nœuds. Quand ce nombre devient grand pour un nœud donné, ceci signifie que ce dernier se trouve dans une zone *« difficile »* d'accès. Une exploration de cette zone est alors favorisée.

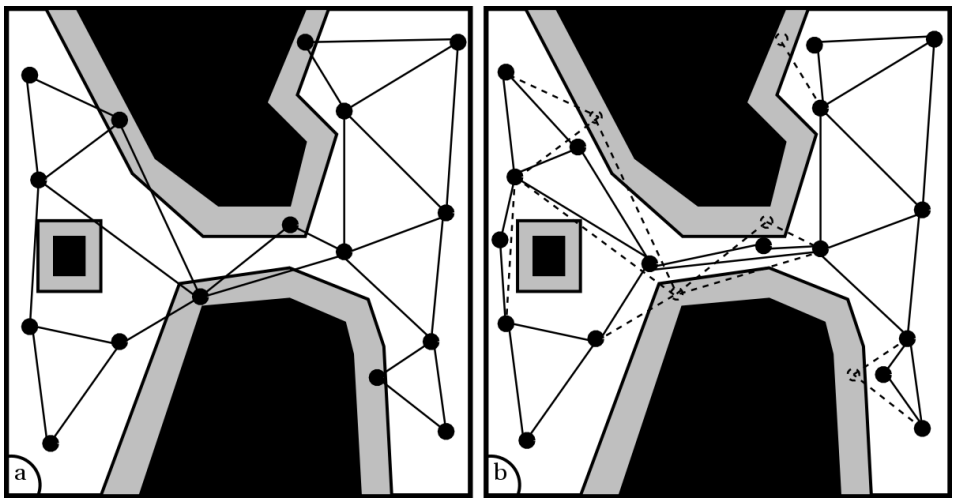


Figure 2.4: a) *Roadmap* calculée en permettant une distance de pénétration dans les obstacles. b) Modification de la *roadmap* en repoussant les configurations non valides.

Dans [Hsu 98, Wilmarth 99a, Wilmarth 99b] la méthode exploite des calculs de distance de pénétration pour calculer en premier temps une *roadmap* dont certains nœuds sont en collision. En effet, uniquement de faibles pénétrations sont permises. La *roadmap* est ensuite modifiée en repoussant progressivement les configurations en collision vers l'espace libre (voir figure 2.4). D'autres méthodes se basant sur le même principe ont été proposées dans la littérature [Ferbach 94, Ferbach 96]. Ce type de méthodes demande une étude géométrique complexe sur les obstacles. Elles sont de plus coûteuses en terme de temps de calcul car elles nécessitent un calcul des distances de pénétration. Elles demandent également un réglage de certains paramètres relatifs aux problèmes traités. Elles sont généralement limitées à des robots de structure simple (free-flying).

Une méthode contournant la complexité de cette étude géométrique est la technique d'échantillonnage gaussien présentée dans [Boor 99]. La technique proposée tire aléatoirement des paires de configurations séparées par une distance selon une distribution gaussienne et ne conserve un échantillon situé dans l'espace libre que lorsque le second échantillon est en collision.

Généralement, les régions les plus difficiles à capturer sont les régions situées entre au moins deux obstacles, celles qui forment des passages étroits (figure 2.5). Cependant, l'échantillonnage gaussien génère des configurations le long des obstacles et non seulement dans les passages étroits. Afin de remédier à cela, un tirage de trois configurations, au lieu d'uniquement deux, distantes de d selon une distribution gaussienne peut être envisagé. Dans ce cas, une configuration libre est conservée si et seulement si les deux autres sont interdites (en collision).

Cette méthode d'échantillonnage gaussien est efficace pour des environnements présentant des passages étroits. Cependant, elle est très sensible au réglage des paramétres de la distribution gaussienne. L'utilisation de cette méthode est généralement limitée au robot de type *« free-flying »*.

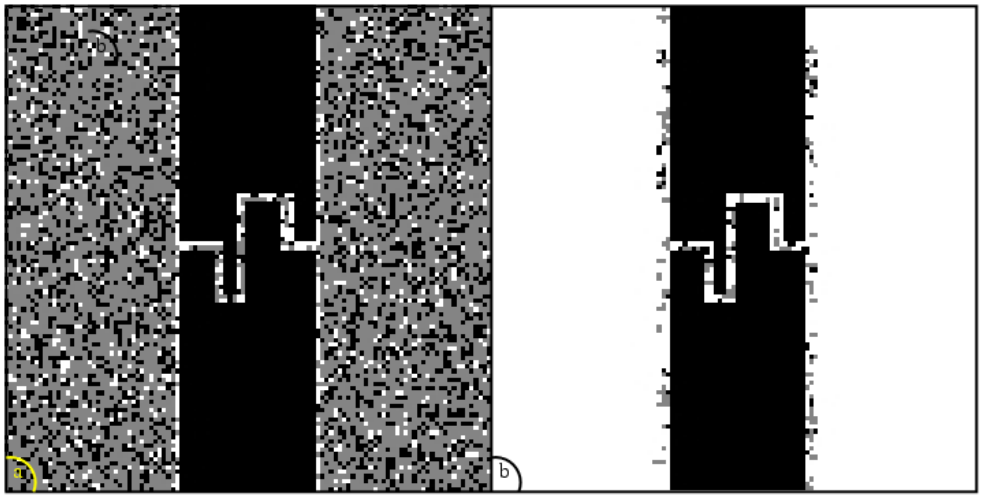


Figure 2.5: a) Échantillonnage Uniforme. b) Échantillonnage Gaussien.

2.3.2 Sélection des nœuds intéressants

D'autres approches visant à minimiser soit la taille de la *roadmap* construite lors de la phase d'apprentissage, soit le nombre d'appels au détecteur de collision lors de la même phase, ont été proposées dans la littérature.

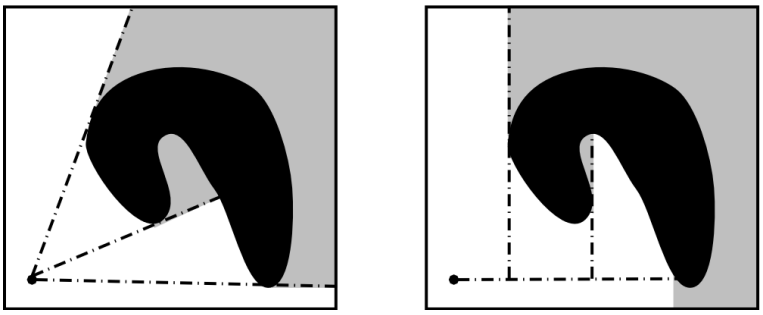


Figure 2.6: Domaines de visibilité d'une configuration pour la méthode locale Linéaire (à gauche) et Manhattan (x puis y) (à droite). Les domaines grisés ne sont pas atteignables.

L'approche, appelée « *Visibility PRM* » [Nissoux 99a, Nissoux 99b, Siméon 00, Laumond 00], combine l'échantillonnage aléatoire avec une notion de visibilité. Cette technique permet de réduire la complexité de la *roadmap* et de contrôler sa construction. L'idée est de guider la couverture aléatoire de l'espace de configurations en exploitant la notion de visibilité de la méthode locale utilisée pour relier deux configurations. Pour un même robot et un même environnement, la forme des domaines de visibilité dépend fortement de la méthode locale choisie [Laumond 00]. Une configuration est dite visible d'une configuration donnée par une méthode locale $\mathcal{L}$ si et seulement si le chemin local $\mathcal{L}$ est sans collision (figure 2.6).

L'algorithme de visibilité ne va garder un nœud tiré aléatoirement que s'il est isolé (i.e. qu'il ne peut pas être relié aux autres nœuds) ou s'il permet de connecter au moins deux composantes connexes distinctes.

La figure 2.7 montre deux *roadmaps* calculées à partir d'une même liste de configurations aléatoires.

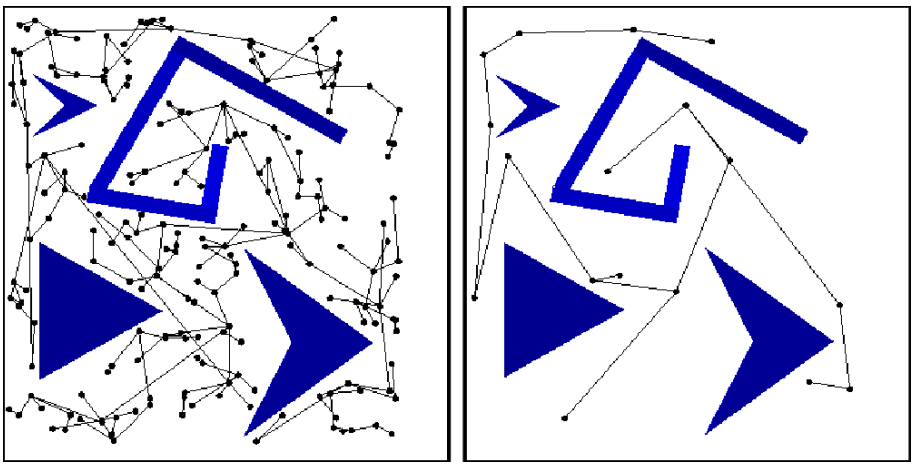


Figure 2.7: Deux graphes issus du même échantillonnage: PRM classique à droite et visibilité PRM à gauche.

La *roadmap* de visibilité (figure de droite) est nettement plus compact que celle générée par la méthode probabiliste classique. La taille réduite de la *roadmap* est un second avantage pour cette approche, elle conduit à un gain significatif en temps de calcul. Il est à noter que cette taille dépend fortement de la méthode locale utilisée ainsi que de l'espace de configuration libre.

2.4 Les méthodes dites « par diffusion »

Les méthodes décrites dans cette section reposent, sur un principe similaire aux méthodes probabilistes classiques. Cependant, les phases d'apprentissage et de recherche sont confondues pour trouver une solution propre à un problème donné.

Ainsi la *roadmap* construite dépend fortement des configurations initiale et finale du problème de planification de mouvement. Pour le même environnement, le même robot et la même méthode locale, ces méthodes ne peuvent souvent pas réutiliser efficacement une *roadmap* construite lors d'une requête préalable pour résoudre un autre problème de planification.

2.4.1 Les méthodes « RRT »

La méthode RRT *« basique »* (*« Rapidly-exploring Random Trees »*), décrite dans [LaValle. 98, LaValle 99a], consiste à faire évoluer un arbre partant de la configuration initiale afin d'atteindre la configuration but. A chaque itération, un nouveau nœud est aléatoirement généré et rajouté à l'arbre s'il n'est pas en collision. La figure 2.8 illustre la construction incrémentale d'un arbre par cette méthode. Le point au centre représente la configuration initiale.

Plusieurs extensions et variantes de cette méthode ont été présentées dans [Kuffner 00, LaValle 00, LaValle 01]. Parmi ces variantes, l'algorithme *« RRT-bidirectionnel »* qui consiste à faire évoluer deux arbres simultanément ayant comme racine respective la configuration initiale et but. Les deux arbres progressent l'un vers l'autre selon une heuristique simple.

La méthode RRT ainsi que ses variantes ont montré des performances meilleures que les méthodes

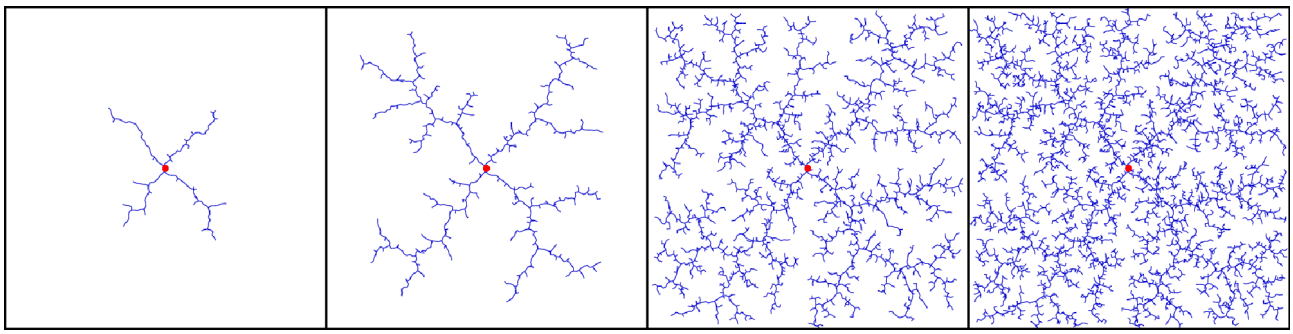


Figure 2.8: Évolution d'un arbre couvrant l'espace libre par la méthode RRT.

probabilistes classiques lorsque les positions initiales ou finales sont fortement contraintes. Par contre elles sont peu efficaces pour trouver les passages étroits, et moins performantes sur des problèmes type labyrinthe que les méthodes globales. Elles sont généralement utilisées pour résoudre des problèmes d'assemblages mécaniques où la distance aux obstacles reste très faible autour de la trajectoire solution.

2.4.2 L'algorithme « Fil d'Ariane »

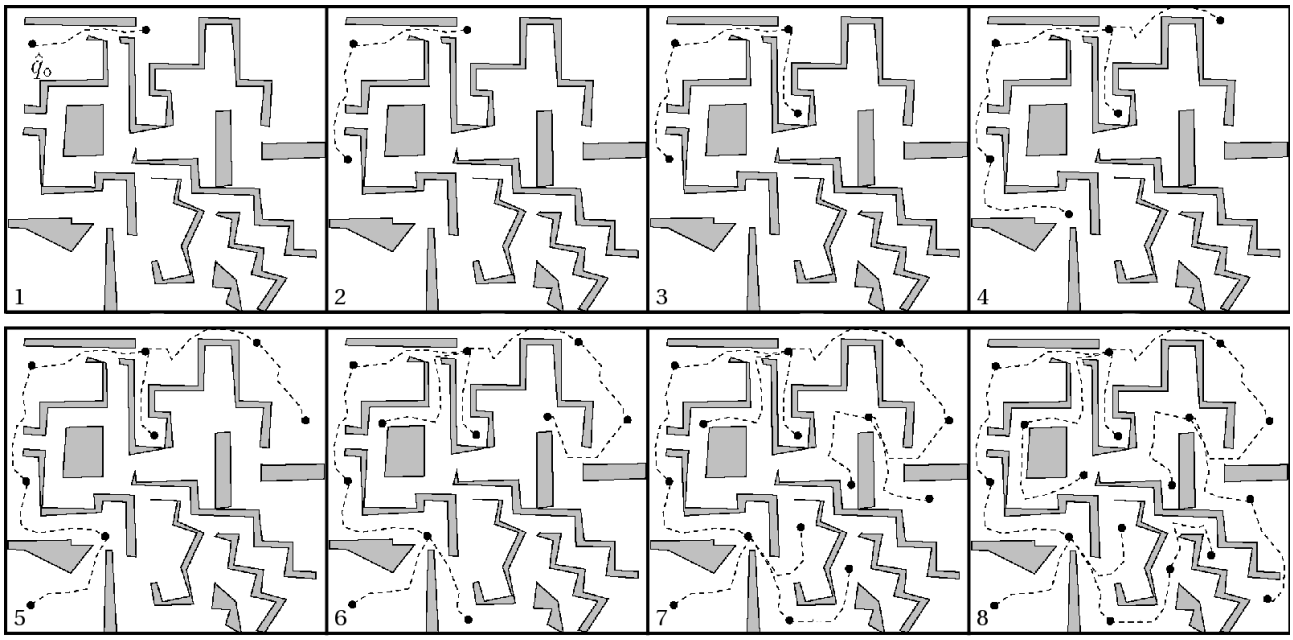


Figure 2.9: Évolution des balises dans l'espace libre tiré de [Ahuactzin 94].

L'algorithme *« Fil d'Ariane »*, décrit dans [Bessière 93, Ahuactzin 94, Mazer 98], utilise une méthode d'optimisation des plans (algorithmes génétiques [Ahuactzin 92]) pour résoudre des problèmes de planification de trajectoire. Les auteurs ne se placent pas, comme dans la plupart des planificateurs, dans l'espace des configurations mais plutôt dans l'espace des commandes. Ils définissent un plan comme un ensemble de commandes pour les actionneurs. Pour un robot holonome, un plan peut être défini par une séquence d'action élémentaires du type :

Tourner de θ_1, Avancer de x_1, Tourner de θ_2, Reculer de x_2, etc ...

Pour certains robots (par exemple robot avec remorque), il peut être délicat de trouver une méthode locale qui permette de définir un mouvement entre deux configurations aléatoires qui respecte la cinématique du robot. Si on se place dans l'espace des commandes, on ne cherche pas forcément à atteindre une configuration précise. La configuration finale sera calculée par l'enchaînement des commandes qui est un calcul beaucoup plus simple.

Ainsi l'algorithme du fil d'Ariane remplace l'utilisation d'une méthode locale par une optimisation sur les plans possibles pour trouver une configuration finale proche à ϵ près de la configuration but.

Pour pallier le problème des minima locaux de l'optimisation des plans, les auteurs construisent un arbre de plans permettant l'exploration de tout l'espace libre. Cet arbre est obtenu en posant des balises dans l'espace accessible du robot. Du point de vue implémentation, l'algorithme *« Fil d'Ariane »* est composé de deux sous algorithmes *SEARCH* et *EXPLORE*. Le premier collecte des informations sur l'espace accessible depuis la position initiale en posant des balises dans l'espace de recherche et en mémorisant les chemins entre ces balises et la position initiale.

Le second essaie de placer de nouvelles balises aussi loin que possible de celles déjà placées. La figure 2.9 illustre l'évolution des balises placées par l'algorithme. EXPLORE fait un pavage progressif de l'espace des configurations assurant la complétude de l'algorithme.

2.5 Les chaînes cinématiques fermées

Les chaînes cinématiques fermées apparaissent dans différentes classes de problèmes étudiées en robotique: mécanismes avec boucles cinématiques (robots parallèles), manipulation coordonnée de plusieurs robots (par exemple le cas de plusieurs manipulateurs coopérant pour déplacer un objet), ou encore en modélisation moléculaire (déterminer les conformations énergétiquement stables pour une molécule). Ce type de mécanisme est composé généralement d'un nombre élevé de degrés de liberté ce qui requiert l'utilisation des techniques probabilistes.

L'analyse des chaînes cinématiques fermées est exprimé généralement en tant que problème de cinématique inverse et formulé comme un système d'équations non-linéaires. L'étude de ce type de mécanisme est encore aujourd'hui un sujet actif de recherche et il n'existe pas une solution satisfaisante pour le cas général. Plusieurs méthodes pour résoudre ces systèmes d'équations non-linéaires ont été présentées dans la littérature [Nielsen 97]. Cependant, ces solutions ne possèdent pas les caractéristiques souhaitables (généricité, robustesse, efficacité,...) pour être intégrées dans les techniques probabilistes pour la planification de mouvement.

Alors que pour les chaînes cinématiques ouvertes, la génération aléatoire de configurations est une étape trivial, pour les mécanismes avec des chaînes fermées cette étape nécessite l'utilisation de méthodes plus complexes. En effet l'ensemble des configurations où une chaîne cinématique est fermée est une sous variété de l'espace des configurations. Ainsi la dimension des paramètres utiles est inférieure à celle de l'espace de configuration. La probabilité de tirer aléatoirement une configuration dans une sous-variété de l'espace de configuration est nulle. Il faut donc une méthode pour se ramener à cette sous-variété et donc recalculer des degrés de liberté.

De plus, les contraintes de fermeture doivent être respectées le long d'une trajectoire, ce qui ajoute une difficulté dans le calcul des chemins locaux.

En dépit de l'intérêt croissant porté à ce type de mécanisme, peu de travaux concernant la planification de mouvement pour les chaînes cinématiques fermées ont été présentés dans la littérature [LaValle 99b, Yakey 99, Han 00, Cortés 02].

2.5.1 La fermeture de la chaîne cinématique

Dans [LaValle 99b, Yakey 99], les auteurs combinent la technique d'échantillonnage aléatoire avec une descente de gradient permettant de réduire une fonction d'erreur qui représente les contraintes de fermeture de la chaîne cinématique. Cette approche souffre, comme la majorité des méthodes utilisant la descente de gradient, de problème de convergence dès qu'il s'agit d'un mécanisme complexe.

La méthode présentée dans [Han 00] consiste à casser la boucle formée par le système en deux chaînes : *active* et *passive*. La configuration de la chaîne active est générée par un tirage aléatoire, alors que pour la chaîne passive une cinématique inverse est utilisée afin d'assurer la fermeture de la chaîne globale. Concernant la méthode locale connectant deux configurations, les auteurs proposent de planifier uniquement les degrés de liberté de la partie active, et d'utiliser la fonction de cinématique inverse pour en déduire la partie passive. Cette méthode sera utilisée tout au long de cette thèse.

2.5.2 La méthode d'échantillonnage.

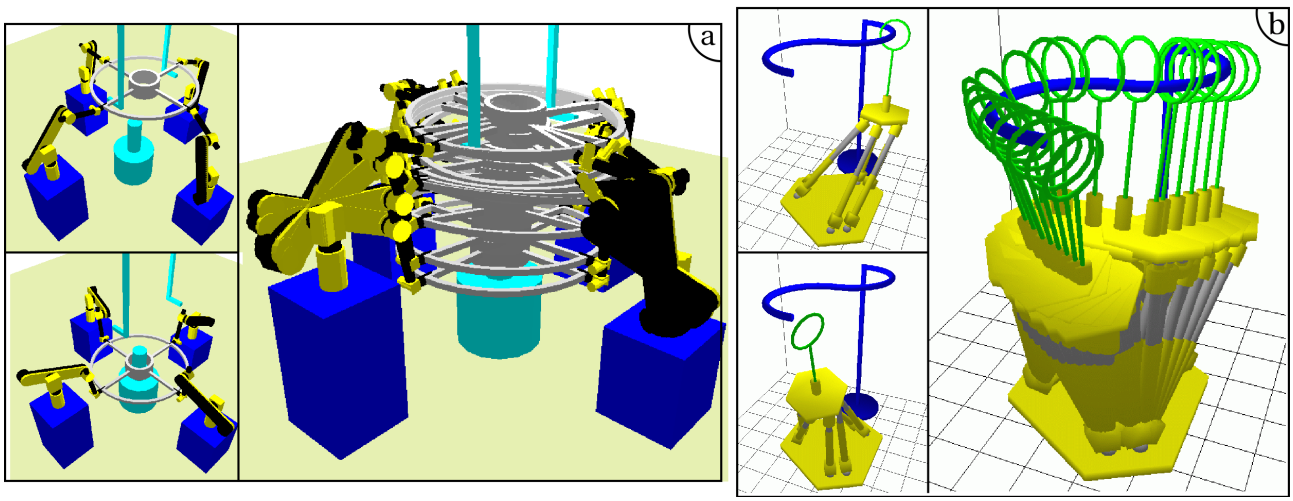


Figure 2.10: Exemples de problèmes de planification de mouvement pour des mécanismes fermés [Cortés 03]. (a) coopération de bras manipulateur pour déplacer un objet. (b) robot parallèle : la plate-forme de Stewart.

Le principal inconvénient qui existe avec l'utilisation de chaînes *passives* concerne la manière de générer les nœuds : lorsque l'organe terminal de la chaîne active (dont la configuration est générée aléatoirement) n'est pas atteignable par la chaîne passive, une configuration fermée du système ne peut pas être obtenue. Cet inconvénient se répercute sur les performances de la méthode. Dans [Cortés 02], les auteurs reprennent la même approche de [Han 00], en modifiant la façon du tirage aléatoire de la chaîne active. Ils proposent un algorithme, appelé « *Random Loop Generator* » (*RLG*), permettant de

guider l'organe terminal de la chaîne active vers une approximation conservative de l'espace atteignable par la chaîne passive afin d'augmenter la probabilité de générer des configurations valides (fermées).

2.5.3 Les « kinematic roadmaps »

Pour des robots complexes ou un ensemble de robots qui saisissent un même objet (figure 2.10.a), la complexité de ce calcul peut rester encore importante, mais surtout le risque d'auto-collision ou d'échec de la méthode de cinématique inverse est assez fort.

Ainsi quand une méthode locale doit être validée pour un tel robot, il faut à la fois prendre en compte les collisions avec l'environnement et les problèmes de fermeture de chaînes et d'auto-collision. Or il se trouve que ces problèmes de fermeture de chaînes cinématique sont indépendants de l'environnement. Il pourrait donc être intéressant de séparer le test de la méthode locale en deux parties, une première permettant de vérifier la validité du chemin par rapport au robot seul et une autre pour vérifier la validité du chemin par rapport à l'environnement.

Généralement les robots mobiles peuvent être séparés en deux parties, une partie qui fait les déplacements dans l'espace, que l'on appellera la base du robot[1], et une partie constituée par un ensemble de chaînes cinématique relié à cette base[2]. Dans ce cas il est souvent possible de séparer les mouvements internes des mouvements de la base du robot.

Nous allons expliquer que, quand cette séparation est possible, un mouvement interne au robot peut être reproduit sur diverses trajectoires construites pour la base du robot. Cette idée à été développé par Han et Amato avec l'introduction des *« kinematic roadmaps »* [Han 00].

La résolution d'un tel problème de planification de mouvement utilise deux roadmaps différentes, l'une pour rechercher les mouvements internes du robot et l'autre pour pouvoir rechercher les mouvements possibles dans l'environnement.

La *« kinematic roadmap »* va capturer la topologie des mouvements internes. Pour cela elle sera construite indépendamment de l'environnement et la base du robot sera fixe. Elle sera donc réutilisable pour des environnements différents.

La roadmap de déplacement quant à elle, va utiliser les nœuds de la *« kinematic roadmap »* en tirant aléatoirement la configuration de la base du robot. Deux nœuds de cette roadmap sont connectables seulement si les nœuds correspondants dans la *« kinematic roadmap »* sont connectés. Il n'y a plus alors besoin de faire le calcul de fermeture de chaîne cinématique, ni de détecter les auto-collisions il suffit juste de vérifier la connexité de la *« kinematic roadmap »* et de tester les collisions avec l'environnement pour savoir si le chemin est valide.

L'inconvénient de cette méthode est qu'il faut avoir une base du robot séparée de la chaîne cinématique à fermer. Or pour les opérations de manipulation, il est possible d'utiliser deux robots qui portent un même objet, dans ce cas nous n'avons pas une base indépendante des chaînes cinématique à fermer.

[1] Pour un robot mobile cela correspond à x, y, θ. Pour un corps libre ce sont les 6 degrés de placement dans l'espace.

[2] Cela peut être des bras qui tiennent un objet, une remorque, des jambes pour un humanoïde, ...

2.6 Les problèmes avec plusieurs robots.

Les problèmes avec plusieurs robots indépendants se trouvent être souvent étonnamment plus complexes que les problèmes avec un robot ayant autant de degrés de libertés que l'ensemble des robots indépendants. Ainsi un problème avec trois robots indépendants dans un environnement un peu contraint sera beaucoup plus délicat que la résolution d'un problème avec trois robots qui portent ensemble une table.

Il existe plusieurs approches de planification de mouvement pour plusieurs robots. Elles sont de deux types : la planification distribuée avec coordination [O'Donnell 89, Qutub 98, Oliver 00] et la planification centralisée [Warren 90, J. Barraquand 92, Svestka 98]. Parmi ces approches nous ne nous intéresserons qu'à celles basées sur les *roadmaps* probabilistes.

2.6.1 Un seul robot ?

L'approche la plus triviale est de concaténer les n robots individuels R_i pour former un robot unique $\mathcal{R}$. Sa configurations sera $\mathcal{C} = (C_1, \ldots, C_n)$ où C_i représente la configuration du robot R_i. Sa méthode locale $\mathcal{L}$ sera aussi la concaténation des méthodes locales L_i des robots R_i, chacune agissant sur la partie C_i de la configuration $\mathcal{C}$ qui lui correspond.

Cette approche est une des rares approches qui assurent une complétude en probabilité en autorisant l'utilisation de robots complexes, mais elle est peu efficace.

En effet la probabilité d'avoir un chemin valide pour le robot $\mathcal{R}$ diminue fortement avec le nombre de robots. En plus l'espace de recherche croit exponentiellement avec le nombre de degrés de liberté. La recherche devient alors très complexe.

Il est important de noter que la recherche d'une solution pour la composition de 2 robots élémentaires peut être plus coûteuse que la recherche d'une solution pour le système de deux robots tenant un objet et ceci même s'il faut faire des calcules de fermeture de chaînes cinématiques (figure 2.11).

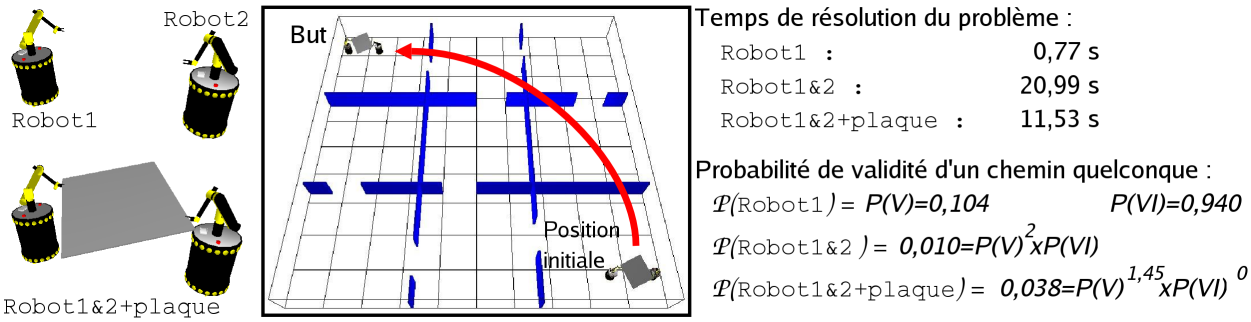


Figure 2.11: Recherche du solution pour passer d'un côté à l'autre de l'environnement dans les cas d'un seul robot, de la concaténation de deux robots et de deux robots portant une plaque. Les temps sont donnés pour une moyenne de 50 recherche et les probabilités sont données à partir de 500000 tirages aléatoires.

Ce résultat expérimental peut sembler étonnant, mais la complexité de résolution d'un problème par des méthodes probabilistes est fonction non seulement de la taille de l'espace de configuration (qui croît exponentiellement avec le nombre de robot) mais aussi de la facilité à relier deux points de cette

espace. Nous allons montrer que dans le cas de robots indépendants, la probabilité de validité d'un chemin tirée aléatoirement dans l'espace des configurations décroît de manière exponentiel avec le carré du nombre de robot. Et que pour les faibles nombres de robots cette décroissance est extrêmement rapide.

Notons par V_i le fait qu'un chemin quelconque du robot R_i soit valide dans l'environnement statique et $P(V_i)$ sa probabilité. De même définissons par $P(VI_{i\,j})$ (Valide Intersection) la probabilité que deux chemins tirés aléatoirement pour les robots R_i et R_j ne s'intersectent pas en un même instant, c'est à dire que les robots R_i et R_j n'entre pas en collision pour une même abscisse curviligne le long de leurs chemins respectifs.

Vu la méthode de construction du robot $\mathcal{R}$, un chemin tiré aléatoirement pour $\mathcal{R}$ est équivalent à n chemins tirés aléatoirement pour les robots R_i. Donc la probabilité $\mathcal{P}$ pour qu'un chemin tiré aléatoirement pour $\mathcal{R}$ soit valide est :

$$
\begin{array}{rlcccl}
\mathcal{P} = & P(V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{1\,n} \cap \ldots \cdots \cap VI_{n-1\,n}) & & & & \\
\mathcal{P} = & P(V_1) \times \frac{P(V_1 \cap V_2)}{P(V_1)} \times \frac{P(V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{1\,n} \cap \ldots \cdots \cap VI_{n-1\,n})}{P(V_1 \cap V_2)} & & & & \\
\mathcal{P} = & P(V_1) \times \frac{P(V_1 \cap V_2)}{P(V_1)} & \times & \ldots & \times & \frac{P(V_1 \cap \cdots \cap V_n)}{P(V_1 \cap \cdots \cap V_{n-1})} \times \\
& \frac{P(V_1 \cap \cdots \cap V_n \cap VI_{1\,2})}{P(V_1 \cap \cdots \cap V_n)} & \times & \ldots & \times & \frac{P(V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{1\,n})}{P(V_1 \cap \cdots \cap V_n \cap \cdots \cap VI_{1\,n-1})} \times \\
& & & \ddots & & \vdots \\
& & & & \times & \frac{P(V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{1\,n} \cap \ldots \cdots \cap VI_{n-1\,n})}{P(V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{1\,n} \cap \ldots \cdots \cap VI_{n-2\,n})} \\
\mathcal{P} = & P(V_1) \times P(V_2/V_1) & \times & \ldots & \times & P(V_n/V_1 \cap \cdots \cap V_{n-1}) \times \\
& P(VI_{1\,2}/V_1 \cap \cdots \cap V_n) & \times & \ldots & \times & P(VI_{1\,n}/V_1 \cap \cdots \cap V_n \cap \cdots \cap VI_{1\,n-1}) \times \\
& & & \ddots & & \vdots \\
& & & & \times & P(VI_{n-1\,n}/V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{n-2\,n})
\end{array}
$$

$$
\begin{array}{rl}
\mathcal{P} = & \left(\prod_{i=1}^{n} P(V_i/V_1 \cap \cdots \cap V_{i-1}) \right) \times \\
& \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} P(VI_{i\,j}/V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{i\,j-1}) \right)
\end{array}
\tag{2.1}
$$

Si nous supposons les robots identiques et les tirages aléatoires indépendant nous avons les simplifications suivantes :

$$
\begin{array}{l}
P(V_i/V_1 \cap \cdots \cap V_{i-1}) = P(V_i) = P(V) \\
P(VI_{i\,j}/V_1 \cap \cdots \cap V_n \cap VI_{1\,2} \cap \cdots \cap VI_{i\,j-1}) = \\
\qquad P(VI_{i\,j}/V_i \cap V_j \cap VI_{i\,i+1} \cap \cdots \cap VI_{i\,n} \cap VI_{j\,j+1} \cap \cdots \cap VI_{j\,n})
\end{array}
$$

Avec $P(V)$ probabilité de validité d'un chemin pour un robot R quelconque.

Nous allons faire une simplification supplémentaire :

$$
P(VI_{i\,j}/V_i \cap V_j \cap VI_{i\,i+1} \cap \cdots \cap VI_{i\,n} \cap VI_{j\,j+1} \cap \cdots \cap VI_{j\,n}) \approx P(VI_{i\,j}) = P(VI)
$$

Même si $P(VI_{i\,j}/V_i \cap V_j \cap VI_{i\,i+1} \cap \cdots \cap VI_{i\,n} \cap VI_{j\,j+1} \cap \cdots \cap VI_{j\,n})$ dépend de V_i, V_j et des $VI_{l\,k}$ précédents nous faisont l'hypothèse que cette probabilité est constante de valeur $P(VI)$.

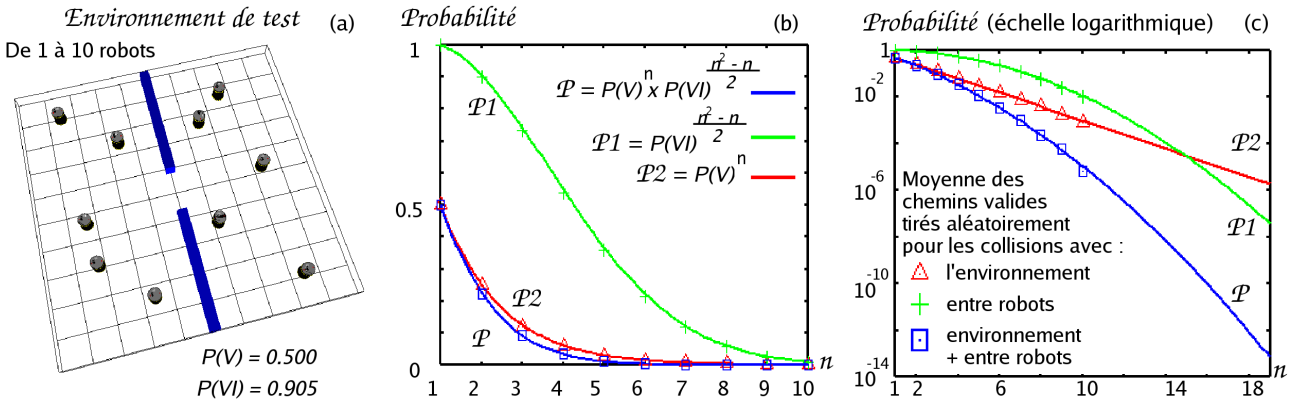


Figure 2.12: Évolution de l'estimation de la probabilité de validité d'un chemin de composition de robots autonomes en fonction du nombre de robot. Pour l'exemple (a) nous avons estimé $P(V)$ et $P(VI)$. (b) et (c) montrent l'évolution théorique et expérimentale de ces probabilités, avec 1 million de tirages aléatoire de chemins pour chaque point expérimental. Nous pouvons constater la corrélation des courbes théoriques et expérimentales.

Cela est vérifié expérimentalement (figure 2.12), et correspond au fait que deux chemins valides dans l'environnement ont plus de chance de s'intersecter vu que leur trajectoire est dans un espace restreint, et moins de chance vu qu'ils sont probablement plus courts. Il y a compensation des effets qui rendent cette probabilité assez stable quel que soit le nombre de trajectoire que les robots n'intersectent pas ($VI_{i\,i+1} \cap \cdots \cap VI_{j\,n}$).

Nous obtenons donc : $\mathcal{P} = P(V)^n \times P(VI)^{\frac{n^2-n}{2}}$

$P(V)^n$ représente les collisions avec l'environnement et $P(VI)^{\frac{n^2-n}{2}}$ les collisions entre robots. Cette probabilité décroît exponentiellement au carré du nombre de robots (figure 2.12).

Toutefois, pour des environnement réaliste, le volume occupé par les obstacles statiques est bien supérieur au volume d'un robot individuel, ainsi la probabilité $P(V)$ est bien inférieur à la probabilité $P(VI)$. Pour un faible nombre de robot, c'est le facteur $P(V)^n$ qui est prédominant (figure 2.12).

Ainsi, pour un faible nombre de robot, le problème de validation proviendra essentiellement de la validation des robots individuels.

Cela permet de montrer l'intérêt de rechercher une solution dans le graphe produit cartésien des roadmaps de chaque robot R_i [Svestka 98]. Par construction, ce graphe permet de connaître les arcs valides sans prendre en compte les interactions entre robots ce qui est justement représenté par le facteur $P(V)^n$. Un arc quelconque de ce graphe aura une probabilité d'être valide de l'ordre de $P(VI)^{\frac{n^2-n}{2}}$ (figure 2.12).

D'un autre coté, le calcul de la probabilité de validité d'un chemin quelconque pour un robot ayant n corps mobiles peut être aussi obtenue par l'équation 2.1 en considérant les corps du robot comme des sous robots. Par contre les simplifications faites ci-dessus ne sont plus valides.

Comme les corps sont proches les uns des autres, si un corps passe il est probable que les corps qui sont proches de celui-ci passent aussi. Ainsi $P(V_i/V_1 \cap \cdots \cap V_{i-1}) > P(V_i)$.

Cette relation dépendant fortement de la géométrie du problème et de la visibilité de la méthode locale employée, il ne nous est pas possible de déterminer de manière générale cette relation.

Dans l'exemple de la figure 2.11 nous avons obtenu expérimentalement pour les deux robots tenant un plateau $P(V_2/V_1) = P(V)^{0.45} > P(V)$

De plus, le facteur $P(VI)^{\frac{n^2-n}{2}}$ représente les auto-collisions du robot composé. Or lors de l'élaboration du robot il est possible de définir des limites pour les degrés de libertés qui réduisent le risque de collision entre les robots individuels. Dans le cas la figure 2.11 les limites sont telles que les robots individuels ne peuvent pas rentrer en collision avec le plateau. Nous avons donc $P(VI_{1\,2}/V_1 \cap V_2) = 1 = P(VI)^0 > P(VI)$

Ainsi, à degré de liberté équivalent, il est bien plus difficile de travailler avec une concaténation de robots indépendants qu'avec un seul robot plus complexe.

Nous choisirons de développer des roadmaps pour des robots isolés ou des robots liés physiquement à d'autres robots ou objets.

2.6.2 La coordination.

Une autre possibilité souvent utilisée est de planifier les trajectoires indépendamment pour chaque robot et de les coordonner. Cette méthode est beaucoup plus rapide que la précédente. Par contre elle ne permet pas d'avoir une complétude.

Pour parer à cela, il est possible d'utiliser des techniques issues de la coordination de trajectoires [R. Alami 98]. C'est une approche distribuée du problème de coordination. Chaque robot planifie indépendamment sa trajectoire puis en insère une partie dans le plan des autres robots. Les robots calculent des contraintes de synchronisation. Ces contraintes sont alors mises en commun pour détecter d'éventuels cycles. S'il n'y a pas de cycle l'insertion s'est bien passée et le plan global est valide.

En cas de conflit, le plan n'est pas insérable il faut soit re-planifier, soit attendre une opportunité (que les robots qui gênent se déplacent). Les deux possibilités ont été prises en compte par Sam Qutub [Qutub 98]. Dans le premier cas une déformation de la trajectoire est envisagée, mais on se met aussi en attente des robots qui sont gênants tant qu'aucune solution n'est trouvée.

De plus l'algorithme est capable de détecter des cycles d'attente sur l'ensemble des robots. Si un tel cycle apparaît il n'est alors plus possible de trouver une solution de manière indépendante, il faut alors appeler un planificateur centralisé. Celui-ci prendra en compte l'ensemble des robots pris dans le cycle [S. Qutub 97].

Il se trouve après expérimentation que le nombre de robots qui sont pris dans des cycles est souvent assez faible. Cela permet d'éviter une trop forte complexité.

Nous avons vu dans cette section plusieurs algorithmes permettant de trouver une solution au problème de planifications de trajectoires. Les méthodes à diffusion permettent notamment de s'extraire de situations très contraintes mais elles ont des difficultés à trouver les passages étroits. Les algorithmes classiques peuvent compenser ces défauts. En fait il n'existe pas à l'heure actuelle un

algorithme qui ne peut pas être mis en défaut sur un environnement particulier. Par contre l'ensemble des algorithmes présentés ici permettent d'aborder un panel de problème assez large. Il peut sembler avantageux de pouvoir combiner ces techniques pour rechercher la solution à un problème, nous verrons dans le chapitre 4 comment nous avons procédé.

Le problème de la planification avec plusieurs robots reste un problème délicat, l'idée que nous allons développer est d'utiliser une composition de robots dès que ceux-ci seront liés physiquement entre eux pour avoir ces notions de proximité qui permettent de diminuer les probabilités d'échec et sinon, de faire de la coordination.

Mais cette coordination ne se fera pas sur des trajectoires, mais sur des *roadmaps* qui pourront être enrichies en prenant en compte les autres robots. Cela reprend les idées des travaux de Svestka et Overmars [Svestka 98].

2.7 Le domaine de la manipulation.

La planification de mouvement d'un robot évoluant parmi des obstacles fixes ainsi que plusieurs variétés du problème basique ont été largement étudiées au long des vingt dernières années. Cependant, les recherches traitant du problème de planification de tâches de manipulation sont assez récentes.

Le premier article s'attaquant à la problématique de planification de tâches de manipulation est celui de Wilfong [Wilfong 88]. Il s'intéresse au problème de planification de mouvement pour un robot en présence d'objets déplaçables dans des environnements polygonaux et convexes. Le robot peut éventuellement déplacer les objets afin d'atteindre son but. Le placement final des objets déplaçables n'apparaît pas forcément de façon explicite dans la définition du problème. L'auteur considère un polygone convexe en translation dans un environnement 2D. Le robot *"prend"* un objet lorsque l'une de ces arêtes coïncide avec l'une des arêtes de l'objet.

Dans le cas où le placement final des objets déplaçables n'est pas spécifié, Wilfong montre que le problème est *NP dur* [3] (i.e : il peut être résolu par un algorithme de complexité en temps polynomial sur une machine non déterministe [4]). Par ailleurs, si le placement final des objets est complètement spécifié, il montre qu'il s'agit d'un problème *P-ESPACE dur* (i.e : il peut être résolu par un algorithme de complexité en espace mémoire polynomiale sur une machine non déterministe). Dans le cas particulier d'environnement polygonal avec un seul objet déplaçable, il a montré l'existence d'un algorithme en $O(n^3 \log^2 n)$, où n est le nombre de sommets de tous les objets présents dans l'environnement.

Dans la même période, Laumond et Alami ont proposé un algorithme en $O(n^4)$ pour la résolution d'un problème similaire [Laumond 89] : le cas d'un robot et d'un objet circulaire évoluant dans un

[3] Un problème est dit *dur* pour une classe si tous les problèmes de la classe peuvent s'y ramener par une transformation polynomiale.

[4] Une machine qui disposerait d'instructions de choix pour dédoubler le programme en programme parallèle. La complexité d'un algorithme non déterministe correspond au temps pris par un algorithme qui aurait toujours effectué le bon choix.

environnement polygonal. Le nombre de prises de l'objet est supposé infini. Dans cet article, les auteurs introduisent la notion d'espace des actionneurs comme étant un sous-espace de *CS* (espace des configurations du robot + objet) associé aux paramètres de configuration sur lesquels il est possible d'agir directement.

2.7.1 Formulation du problème

Dans [Alami 89] et [Alami 94], les auteurs proposent une approche originale pour la planification de tâches de manipulation. Cette approche est basée sur une formulation géométrique du problème. Ils considèrent qu'un chemin de manipulation est un chemin sans collision contraint dans l'espace des configurations composite ($CS = CR \times CO_1 \times ... \times CO_m$, où CR est l'espace de configurations du robot et CO_i est l'espace de configurations de l'objet O_i et de son attachement § 2.8.4). Deux types de contraintes sont introduites : *les contraintes de placement* et *les contraintes de mouvement* des objets déplaçables.

A partir de ces deux contraintes, deux sous-espaces de l'espace de configurations composite sont introduits : l'espace *GRASP* et l'espace *PLACEMENT*. Le premier correspond au sous-espace dans lequel le robot se déplace solidairement avec l'objet déplaçable manipulé sans entrer en collision avec les obstacles ainsi que les autres objets. Alors que dans le second, le robot se déplace seul sans entrer en collision ni avec les obstacles, ni avec les objets déplaçables qui occupent une configuration stable. Le problème de planification de tâches de manipulation apparaît ainsi comme la recherche d'un chemin contraint dans les différentes composantes connexes du sous-espace « *Grasp* ∩ *Placement* » et entre elles.

La plupart des travaux de manipulation se placent dans le cas discret c'est-à-dire qu'ils supposent qu'il existe un nombre fixé de positions de prises de l'objet par le robot et un nombre fixé de positions de pose de l'objet sur le sol. Dans ce cas « *Grasp* ∩ *Placement* » est constitué d'un ensemble discret de configurations. Le but sera de chercher ces configurations et de les relier entre elles par des chemins de *Transfert* dans *GRASP* et des chemins de *Transit* dans *PLACEMENT*.

2.7.2 Les prises et poses continues.

Dans le cas où le planificateur doit prendre en compte un ensemble continu de positions de saisie et/ou un ensemble continu de positions de pose, *Grasp* ∩ *Placement* n'est plus réduit à un ensemble fini de configurations, mais possède une topologie propre.

Grasp ∩ *Placement* est l'espace qui permet de faire le lien entre les chemins de « *Transfert* » et les chemins de « *Transit* ». Il est donc primordial de connaître sa topologie pour pouvoir savoir où saisir l'objet et savoir où le transporter.

Pour cela nous pourrons utiliser une roadmap. A cette roadmap nous allons associer une méthode locale qui portera à la fois sur le robot, l'objet et la configuration de la prise :

- Le robot doit suivre sa méthode locale (utilisée pour les mouvements de « *Transit* »).

- L'objet doit se déplacer tant qu'il est dans une position de placement stable. Ses mouvements peuvent être quelconques tant qu'ils sont continus.
- La configuration de la prise doit varier de manière continue.
- Une condition de fermeture de chaîne cinématique (§ 2.5) doit être appliquée pour pouvoir saisir l'objet à tout moment.

Cette méthode locale possède une *« propriété de réduction »* [J.-P. Laumond 94]. C'est-à-dire que tout chemin utilisant cette méthode locale peut-être transformé en une succession finie de chemins de *Transit* et de *Transfert*. Autrement dit un chemin dans $Grasp \cap Placement$ peut être utilisé pour trouver la solution du problème.

Il ne sert à rien de chercher à connecter des nœuds d'une même composante connexe de $Grasp \cap Placement$ par un chemin de *Transit* ou de *Transfert* puisque nous savons que nous pouvons obtenir un chemin solution par une transformation a posteriori des chemins de $Grasp \cap Placement$.

Pour les problèmes de prises et de poses continues, les composantes connexes de « $Grasp \cap Placement$ » correspondent aux nœuds isolés du cas discret.

« $Grasp \cap Placement$ » est très intéressant car il n'y a pas de degrés de libertés figés (pas d'objet ou de position de prise fixés). Contrairement au *Transit* ou au *Transfert* qui ont une infinité de composantes connexe qui ne peuvent pas être liées entre elles, $Grasp \cap Placement$ peut avoir une seule composante connexe. Il est donc assez facile d'étudier la topologie de $Grasp \cap Placement$.

2.7.3 Le graphe de manipulation.

Il existe au moins trois types de chemins qu'il est intéressant de rechercher dans CS :

- Les chemins de « *Transit* » (dans $PLACEMENT$) : Le robot bouge tout seul.
- Les chemins de « *Transfert* » (dans $GRASP$) : Le robot transporte l'objet avec une position de prise fixée.
- La méthode locale de ($Grasp \cap Placement$) : Le robot peut déplacer l'objet au sol tout en changeant de prise de manière continue.

Définition 5 *Le* graphe de manipulation *[Sahbani 03] représente l'espace de recherche utilisé pour trouver un plan solution au problème de manipulation. Quand il n'y a qu'un seul robot et un seul objet il est constitué de la roadmap de* $Grasp \cap Placement$ *à laquelle sont rajoutés des arcs de Transit et de Transfert.*

Un arc de Transit doit être relié à deux nœuds de $Grasp \cap Placement$ *ayant la même configuration pour l'objet. Il doit être validé en prenant en compte l'objet à cette configuration.*

Un arc de Transfert doit être relié à deux nœuds de $Grasp \cap Placement$ *ayant la même configuration de prise.*

Si les positions initiales et finales du robot et de l'objet sont dans la même composante connexe du graphe de manipulation, alors il existe une solution. Il « suffira » d'extraire la solution du graphe et de l'optimiser. Nous retrouvons ici le schéma en trois phase du PRM (§ 2.2).

Remarque :
Le problème de la manipulation avec plusieurs robots et objets n'a pas encore été introduit dans le cadre des positions continues de prise et de pose. L'utilisation du graphe de manipulation devient plus délicate. En effet chaque robot peut être en *Transit*, *Transfert*, *Grasp* ∩ *Placement* et dans tous ses déplacements il doit prendre en compte tous les autres robots ou objets pour valider son mouvement. Nous verrons une nouvelle structure qui va permettre de généraliser la notion de graphe de manipulation au cas de plusieurs robots (§ 3.4).

2.8 Les planificateurs pour la manipulation.

2.8.1 Les premiers travaux de manipulation.

Dans [Alami 89] et [Alami 94], une fois les sous-espaces *GRASP* et *PLACEMENT* spécifiés, les auteurs se placent dans le cas d'un ensemble fini de prises et de poses pour plusieurs objets déplaçables. Ils présentent un premier algorithme permettant la résolution d'une instance du problème général, à savoir le cas d'ensemble discret de poses et de prises.

L'algorithme présenté est composé de deux modules : un planificateur de tâches et un planificateur de mouvement. Le planificateur de tâches génère les différentes composantes connexes de *Grasp* ∩ *Placement*.

Dans ce cas particulier, elles sont réduites à des simples nœuds du graphe de manipulation. Le nombre de nœuds du graphe de manipulation est alors égal au produit du nombre de prises possibles pour l'ensemble des objets à déplacer et du nombre de poses possible de ces objets. Le planificateur de mouvement est utilisé pour connecter les différents nœuds générés par le premier module. La figure 2.13, tirée de cet article, montre un exemple de problème de manipulation résolu par cet algorithme.

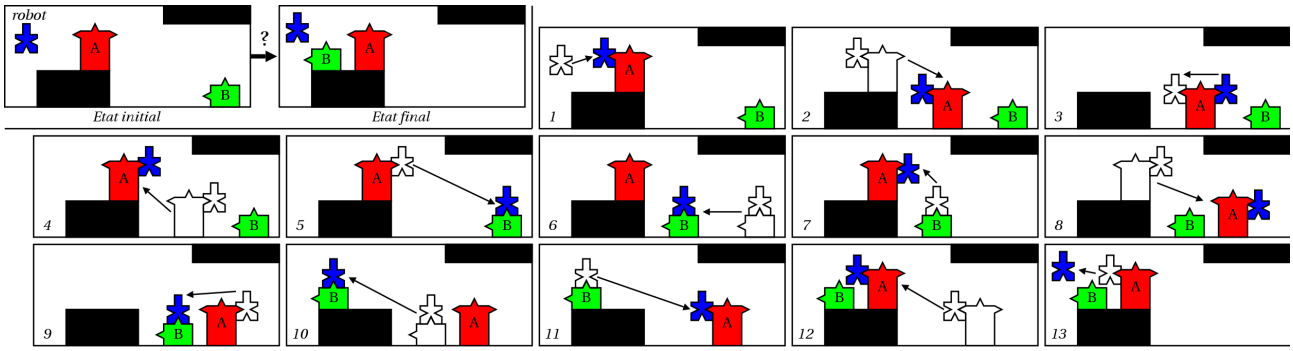


Figure 2.13: Un exemple de problème de manipulation résolu dans [Alami 94] : Pour chaque objet un ensemble de 4 placements stables est défini. Le robot peut « prendre » les objets A et B selon respectivement 3 et 2 prises discrètes définies comme données du problème.

2.8.2 Extension à la manipulation multi-robots.

La même formulation géométrique a été adoptée par Koga et *al.* dans [Koga 94], mais avec une approche un peu différente adaptée aux problèmes de coopération de plusieurs bras manipulateurs pour effectuer une tâche de manipulation (figure 2.14).

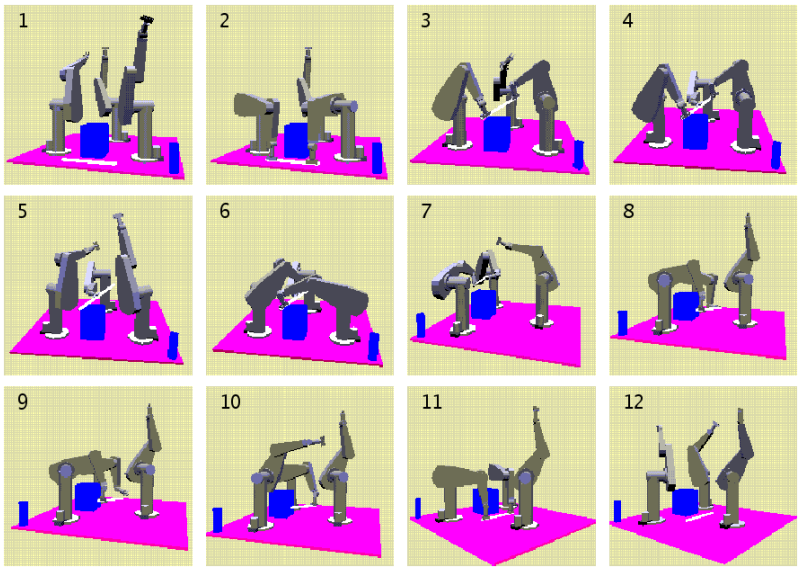


Figure 2.14: Un exemple de coopération entre trois bras manipulateurs de type PUMA 560 tiré de [Koga 95].

Les auteurs proposent une méthode incrémentale pour la construction du graphe de manipulation. Ils se placent dans le cas d'un ensemble fini de prises. Ils définissent pour chaque bras manipulateur une position dite « *non gênante* ». Un bras manipulateur qui ne participe pas au mouvement courant doit occuper cette position.

La définition de l'ensemble des positions de poses stables de l'objet déplaçable dans l'environnement est légèrement adaptée au problème traité. En effet, pour changer de prise, l'un des bras manipulateurs tient l'objet alors que l'autre effectue le changement de prise. Ceci introduit une contrainte supplémentaire : un bras manipulateur effectuant une tâche de « *regrasping* » ne peut pas choisir la prise du bras tenant l'objet.

L'approche proposée dans ces deux papiers commence par planifier un chemin sans collision pour l'objet seul utilisant une méthode probabiliste de type *RPP*. Le chemin de l'objet est défini par une liste de configurations adjacentes. La méthode probabiliste utilisée a été modifiée afin qu'à chaque configuration insérée dans le chemin corresponde au moins une prise parmi celles définies comme entrée du problème.

Une fois le chemin de l'objet calculé, les robots suivent la trace des points de prises afin de garder la même prise le long du chemin. Quand une trace ne peut être suivie, un chemin de *Transit* est calculé permettant ainsi le changement de prise et le suivi le chemin de l'objet.

Le principal défaut de cette méthode, hormis l'utilisation de prise discrète, est qu'il faut définir pour chaque robot une position « *non gênante* » ce qui n'est ni souhaitable, ni toujours possible surtout avec des robots mobiles.

2.8.3 Extension de l'algorithme « Fil d'Ariane ».

Dans [Ahuactzin 98] et [Ahuactzin 95], Ahuactzin et *al.* ont présenté une extension de l'algorithme du *"Fil d'Ariane"* permettant de résoudre des problèmes de planification de tâches de manipulation. La figure 2.15, tirée de [Ahuactzin 98] montre un chemin de manipulation produit par leur planificateur. Ils se sont basés sur la structuration de l'espace de recherche introduite dans [Alami 94].

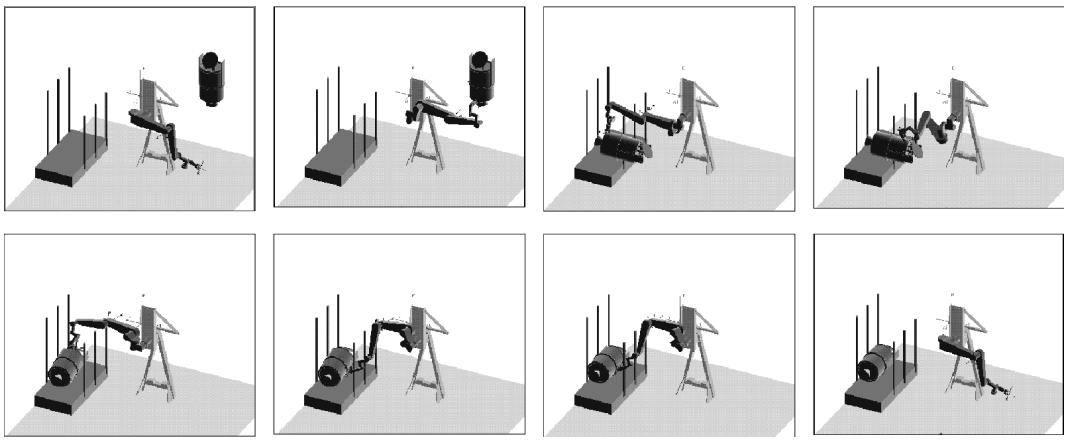


Figure 2.15: Exemple de problème de manipulation résolu dans [Ahuactzin 98].

L'algorithme proposé, *ACA_MANIP*, comporte deux composantes: *E_MANIP* et *S_MANIP*. Le rôle de la première consiste à poser des balises dans le sous-espace $Grasp \cap Placement$ en partant de la configuration initiale. Ces balises représentent les configurations du robot (tenant l'objet) atteignables à partir de la configuration initiale et en favorisant les régions non explorées. Par construction, une nouvelle balise est atteignable d'au moins une des balises déjà placées via un chemin de manipulation. A chaque nouvelle balise placée, *S_MANIP* teste l'existence d'un chemin de *Transfert* connectant la balise courante à la configuration finale. En cas d'échec, *E_MANIP* génère une nouvelle balise.

L'algorithme *S_MANIP* est formulé comme étant un problème d'optimisation qui se présente de la façon suivante. Soit $X \subset \Re^n$, étant donnés un point initial $\hat{x_0} \in X$ et une zone but à atteindre $X_\bullet$, le problème se résume à trouver un chemin $\hat{\rho}$, s'il existe, connectant $\hat{x_0}$ à $X_\bullet$. Notons que $\hat{\rho}(0) = \hat{x_0}$ et que $\hat{\rho}(1) \in X_\bullet$. Ainsi *S_MANIP* consiste à résoudre le problème suivant:

$$\begin{cases} Minimiser \;\; c(\hat{\rho}(1), X_\bullet) \\ \hat{\rho} \in \Omega(X, \hat{x}) \end{cases} \tag{2.2}$$

où $\Omega(X, \hat{x})$ *est l'ensemble des chemins dans* X *partant de* $\hat{x} \in X$.
$c(\hat{\rho}(1), X_\bullet)$ *est une fonction coût estimant la distance entre* $\hat{x}$ *et la région but* $X_\bullet$.

Notons que pour un même environnement, changer la configuration initiale du problème de manipulation demande de reprendre toutes les itérations dès le début. En effet, la configuration initiale sert comme un point de référence pour l'algorithme *E_MANIP*. Les différentes balises placées ainsi que les différents chemins mémorisés dépendent de cette configuration. Cet algorithme sert à répondre à une requête unique de planification de tâches de manipulation du fait que les balises construites

pour cette requête ne peuvent être réutilisées pour une autre. Par ailleurs, ce planificateur traite des problèmes de planification de tâches de manipulation sous des contraintes discrètes. L'ensemble fini de prises et poses est défini par l'utilisateur. Cependant, cet algorithme a été appliqué avec succès sur des problèmes de planification de tâches de manipulation pour des robots redondants pour lesquels la cardinalité du sous-espace $Grasp \cap Placement$ est infinie.

2.8.4 La manipulation avec prise et pose continues.

Sahabani [Sahbani 03] a présenté un algorithme permettant de traiter le cas de la planification de tâches de manipulations pour un robot et un objet avec des positions de prises et de poses continues. Dans ce cas l'espace $Grasp \cap Placement$ est continu et il faut capturer sa topologie (§ 2.7.2).

L'algorithme présenté recherche dans trois sous-espaces de CS pour pouvoir enrichir le graphe de manipulation : $Grasp \cap Placement$, *Transit* ($PLACEMENT$), *Transfert* ($GRASP$). Des roadmaps vont leur être associées afin de capturer la topologie de ces espaces.

L'algorithme prévoit trois extensions pour ces roadmaps :

- Enrichir $Grasp \cap Placement$.
- Relier deux composantes de $Grasp \cap Placement$ non connexes dans le graphe de manipulation par un chemin de *Transfert* (prise fixée).
- Relier deux composantes de $Grasp \cap Placement$ non connexes dans le graphe de manipulation par un chemin de *Transit* (position de l'objet fixée).

Le choix de la méthode d'extension se fait de manière probabiliste avec un niveau de probabilité qui dépend de la visibilité des nœuds de la roadmap de $Grasp \cap Placement$. Plus il sera difficile d'insérer des nœuds avec la méthode de visibilité (§ 2.3.2) dans $Grasp \cap Placement$ plus l'algorithme aura tendance à chercher une solution en dehors de $Grasp \cap Placement$ (méthodes 2 et 3).

La roadmap de *Transit* va être réutilisée pour différentes positions de l'objet. Il faudra la valider par rapport à la position de l'objet correspondant aux nœuds de $Grasp \cap Placement$ à relier. La réutilisation de cette roadmap permet d'utiliser les connaissances sur la topologie de l'environnement qui est acquise au-cours de la recherche et d'avoir un gain en performance.

Dans un premier temps, un RRT (§ 2.4.1) essaie de relier les extrémités des arcs de la roadmap de *Transit* qui permettent de trouver une trajectoire solution sans prendre en compte l'objet mais qui sont invalidés par la présence de cet objet. C'est une étape de correction locale.

Si cette étape échoue, une phase d'apprentissage classique (§ 2.2.1) est utilisée pour chercher une nouvelle solution. Les collisions avec l'objet seront prises en compte tout au long de cette phase.

La roadmap de *Transfert* est peu réutilisable puisqu'il est nécessaire d'avoir la même position de prise tout au long d'un chemin, ce qui n'est pas le cas dans $Grasp \cap Placement$.

Les contraintes *PLACEMENT* et *GRASP* imposent l'utilisation de nœuds particuliers dans la roadmap de *Grasp* ∩ *Placement*. Pour *PLACEMENT* il faut deux nœuds qui correspondent à une même position d'objet et pour *GRASP* il faut deux nœuds qui correspondent à une même position de prise.

Un moyen pour obtenir ces nœuds particuliers est d'avoir la possibilité de générer des nœuds avec une partie de la configuration pré-définie. Cette première possibilité est utilisée comme première étape de la méthode 2 (chemin de *Transfert*).

Par contre pour la méthode 3 (chemin de *Transit*), l'algorithme cherche des positions de pose intéressantes. C'est à dire les positions de pose qui sont communes à deux nœuds de *Grasp* ∩ *Placement* qui ne sont pas dans la même composante connexe. Elles sont appelées positions de *« re-grasping »*.

Une des principales contributions de Sahabani porte justement sur cette recherche de position de *« re-grasping »* qui sont un sous-ensemble souvent très restreint de l'ensemble des positions de pose. Donc difficile à trouver par simple échantillonnage.

2.8.5 Heuristique des positions de « re-grasping »

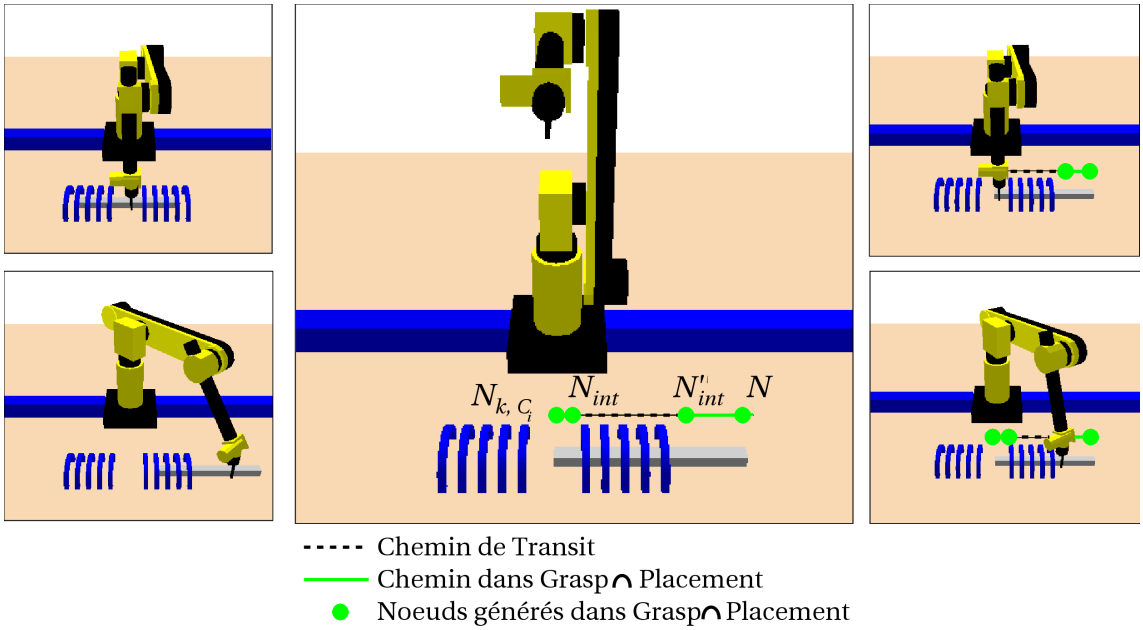


Figure 2.16: Connexion par *« re-grasping »* entre deux nœuds N et N_{k,C_i} de *Grasp* ∩ *Placement*.

Notons par N_{k,C_i} un nœud d'une composante connexe C_i de *Grasp* ∩ *Placement*. L'échec d'une connexion dans *Grasp* ∩ *Placement* entre un nouveau N et N_{k,C_i} peut nous informer sur l'existence d'une opération de *« re-grasping »* possible (un changement de prise permettant d'établir la connexion entre les deux nœuds).

En effet, si le chemin connectant les nœuds N et N_{k,C_i} dans *Grasp* ∩ *Placement* est uniquement valide pour l'objet (le chemin pour le robot étant en collision), cette fonction teste la possibilité de connecter ces deux nœuds en effectuant un changement de prise.

Un parcours de ce chemin est alors effectué afin d'identifier un placement intermédiaire de l'objet p_{int} associé à deux prises intermédiaires g_{int} et g'_{int} (issues des deux configurations du système global sur le chemin juste avant et juste après la collision, si elles existent). L'identification de ce placement intermédiaire est effectuée en exploitant les informations fournies par le détecteur de collision lors du parcours de ce chemin.

Un chemin de *Transit* entre les nœuds $N_{int} = (g_{int}, p_{int})$ et $N'_{int} = (g'_{int}, p_{int})$ est ensuite calculé. Si ce chemin est valide pour le robot et si le chemin dans $Grasp \cap Placement$ connectant N'_{int} et N_{k,C_i} est sans collision, alors N et N_{k,C_i} sont connectés.

Le chemin connectant ces deux nœuds est la séquence : mouvement de $Grasp \cap Placement$ de N à N_{int}, mouvement de *Transit* de N_{int} à N'_{int} et enfin mouvement de $Grasp \cap Placement$ de N'_{int} à N_{k,C_i}.

Finalement, N est considéré comme lié à C_i et sera ajouté au graphe s'il est connecté à au moins une autre composante connexe du graphe de manipulation (critère de visibilité).

La figure 2.16 illustre une opération de *« re-grasping »* déterminée suite à un échec de connexion dans $Grasp \cap Placement$ entre les nœuds N et N_{k,C_i} (à gauche). Le chemin connectant ces deux nœuds dans $Grasp \cap Placement$ est discrétisé en un ensemble de paires (placement, prise). Un test de collision est effectué pour chaque configuration de ce chemin (et donc pour chaque paire (placement, prise)). Le nœud N_{int} correspond à la paire (p_{int}, g_{int}) juste avant collision (collision entre le robot et la cage). Le placement de l'objet déplaçable est alors fixé à p_{int} (placement intermédiaire de l'objet). Le nœud N'_{int} est déterminé en incrémentant le paramètre de la prise (le robot glisse le long de l'objet figé à son placement intermédiaire p_{int}) jusqu'à ce que la configuration du robot soit valide (sans collision).

2.8.6 La transformation des chemins de Grasp ∩ Placement

Une fois que les états initial et final sont connectés à travers le graphe de manipulation, il est possible d'extraire un chemin à travers le graphe. Ce chemin sera constitué de trois mouvement possibles : les mouvements de *Transit* pour une position fixe de l'objet, les mouvements de *Transfert* pour une position de prise donnée, les mouvements dans $Grasp \cap Placement$.

Ces derniers mouvements ne sont pas réalisables physiquement, mais grâce à la propriété de réduction, les chemins calculés dans $Grasp \cap Placement$ sont transformables en une séquence alternée et finie de chemins de *Transit* et de chemins de *Transfert* (comme illustré dans la figure 2.17).

Pour effectuer cette transformation, un algorithme récursif de type dichotomie (*« divide-and-conquer »*) est proposé.

Soit $\mathcal{C}$ un chemin dans $Grasp \cap Placement$ allant de la configuration C_1 à la configuration N_2. L'algorithme essaie de le transformer en un chemin de *Transfert* allant de N_1 à N'_1 suivi d'un chemin de *Transit* de N'_1 à N_2, avec pour N'_1 la configuration de l'objet de N_2 et la configuration de prise de N_1.

Si cette transformation n'est pas possible, le chemin $\mathcal{C}$ est divisé en deux ($\mathcal{C}_1$, $\mathcal{C}_2$) et l'algorithme

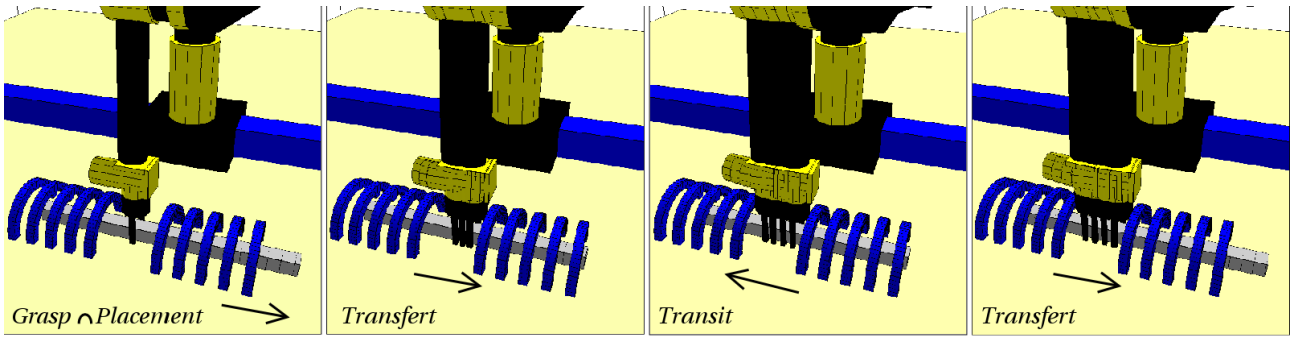


Figure 2.17: Un chemin calculé dans *Grasp* ∩ *Placement* (à gauche) transformé en une séquence de trois chemins admissibles *Transfert*/*Transit*/*Transfert* (droite).

retente récursivement la transformation sur $\mathcal{C}_1$ et $\mathcal{C}_2$.

Une fois l'étape de transformation des chemins calculés dans *Grasp* ∩ *Placement* est achevée, le chemin de manipulation est composé uniquement d'une séquence alternée de chemins de *Transit*/*Transfert*. Ce chemin est admissible et constitue une solution au problème de manipulation posé. Cependant, il est généralement formé par une longue succession de prises et de poses dont la plupart ne sont pas vraiment utiles pour la résolution du problème. Cette longue séquence de prises et de poses est essentiellement due au caractère probabiliste de l'approche.

Une opération d'optimisation va essayer de transformer une séquence de chemins de *Transit*/*Transfert* choisie aléatoirement en un chemin de *Transit* suivi d'un chemin de *Transfert*.

Après cette étape de minimisation des séquences de prises et de poses, une phase de lissage des chemins de *Transit* ainsi que des chemins de *Transfert* est effectuée par des techniques d'optimisation présentées dans la section § 2.2.3.

Conclusion

Nous avons présenté dans ce chapitre les techniques de planifications géométriques et plus précisément celles basées sur des techniques probabilistes. Au fur et à mesure des années ces techniques se sont développées pour s'attaquer à des problèmes de plus en plus complexes.

Chacune de ces techniques peut être très performante pour un domaine donné, mais aucune ne permet de traiter tous les problèmes. Or les problèmes que nous voulons aborder vont faire apparaître de la manipulation, donc des contraintes cinématiques, plusieurs robots, des environnements variés, ...

Pour résoudre ces problèmes il faudra définir un nouvel espace de recherche et de nouvelles techniques d'exploration de l'espace libre pour utiliser ou généraliser les différentes approches vu dans ce chapitre.

Comme nous le verrons dans les deux chapitres suivants, nous avons intégré à notre planificateur « *aSyMov* » la plupart des algorithmes présentés ici ou des méthodes qui généralisent ces algorithmes.

Chapitre 3

L'espace de recherche de la géométrie.

Comme nous l'avons vu dans le chapitre précédent, la planification de mouvement possède un ensemble d'outils performants permettant de résoudre des problèmes géométriques complexes. Notamment les outils de planification probabiliste qui utilisent des roadmaps pour capturer la topologie de l'espace.

Nous voulons résoudre des problèmes de planification de mouvements avec plusieurs robots et objets déplaçables. Nous proposons dans ce chapitre une approche basée sur deux principes :

- La composition des robots : un robot transportant un objet forme un nouveau robot.
- Une roadmap sera associée à chaque type de mouvement de chaque robot ou objet déplaçable (composé ou non).

Pour la manipulation, nous avons remarqué l'utilisation de trois roadmaps pour l'exploration du graphe de manipulation ($Grasp \cap Placement$, *Transfert* et *Transit*), mais avec plusieurs robots et plusieurs objets, ce nombre s'accroît. Il faut donc réviser la structure du graphe de manipulation pour construire un nouvel espace de recherche pouvant mettre en relation tous les types de mouvement pouvant être exécutés.

Dans un premier temps nous allons définir les robots et les objets manipulés, puis voir les roadmaps que nous utiliserons et les liens qui peuvent exister entre elles. Pour finir enfin par la définition d'un nouvel espace de recherche : les *« Graphes des Cinématiques Élémentaires »* (GCEs) [Gravot 03] qui représentera un graphe entre les roadmap et nous indiquera comment résoudre un problème donné.

3.1 Composer et décomposer les robots.

Notre planificateur s'intéresse aux problèmes de manipulation pour un ensemble de robots mobiles et d'objets. Comme nous l'avons dit précédemment, l'approche géométrique va être basée sur les

roadmaps probabilistes. Nous allons donc définir les objets et robots à la fois dans l'espace réel et dans l'espace des configurations. Nous présenterons aussi une méthode de composition des objets et des robots permettant de constituer de nouveaux robots ou objets plus complexes. Cette méthode sera un point fondamental de la manipulation ainsi que de la nouvelle topologie sur l'ensemble des roadmaps que nous allons développer plus loin dans ce chapitre (§ 3.4).

3.1.1 Objets, robots, définitions.

D'un point de vue géométrique, les robots et les objets déplaçables sont similaires. Ils possèdent des corps rigides qui peuvent entrer en collision avec l'environnement, et des articulations qui les relient entre eux. Chaque articulation possède un certain nombre de degrés de liberté (un joint de translation ou de rotation en possède 1, un joint plan ou rotule en possède 3). Une instanciation de ces degrés de liberté représente la configuration d'un robot ou d'un objet.

Dans notre application les robots et les objets déplaçables ont une même représentation, seules les méthodes locales qui leurs sont appliquées changent. Ainsi un robot est capable de mouvement propre. Sa méthode locale (dans une roadmap de mouvement) représentera ses capacités de déplacement. Un objet déplaçable ne pourra pas se mouvoir seul, il n'aura pas de méthodes locales (sa roadmap associée ne possédera pas d'arc). Les objets qui ne sont pas déplaçables seront considérés comme des obstacles statiques qui définiront *l'environnement statique*.

Pour généraliser la notion d'objet et de robots nous parlerons de chaînes cinématiques. Une *chaîne cinématique* est un ensemble d'articulations reliées entre elles auxquelles peuvent être rattachés un ensemble de corps rigides. Une instance des degrés de libertés de ces articulations représente une configuration de la chaîne cinématique.

Nous utiliserons trois types de chaînes cinématiques :

- Les *« chaînes cinématiques élémentaires »* : C'est le niveau le plus fin que l'on va étudier. Elles représentent l'ensemble minimal d'articulations reliant des corps rigides qui définissent l'objet, le robot ou une sous-partie de robot qui sera étudiée indépendamment dans notre problème.
  Une *« chaîne cinématique élémentaire »* ne pourra pas être coupée, ses articulations doivent constamment avoir des valeurs valides.
  Pour pouvoir représenter les corps rigides dans l'espace, les 6 premiers degrés de liberté d'une *« chaîne cinématique élémentaire »* représentent un repère de placement par rapport au référentiel de l'environnement.
- Les *chaînes cinématiques virtuelles* : Elles ont les mêmes propriétés que les *« chaînes cinématiques élémentaires »* mais ne comportent aucun corps rigide. Elles n'ont donc pas besoin des 6 degrés de liberté de placement. Elles sont utilisées pour représenter les attachements entre objets et robots qui peuvent parfois être assez complexes.
  Par exemple un robot qui tiens une barre cylindrique à un attachement défini par un joint plan (pince plate) suivi d'un joint de rotation autour de l'axe de la barre. Toutes les prises sont possibles autour de ce cylindre. Ce n'aurait pas été le cas avec un livre.

- Les *chaînes cinématiques composées* : Elles sont la concaténation de chaînes cinématiques élémentaires, virtuelles ou composées. Elles peuvent donc être divisées.

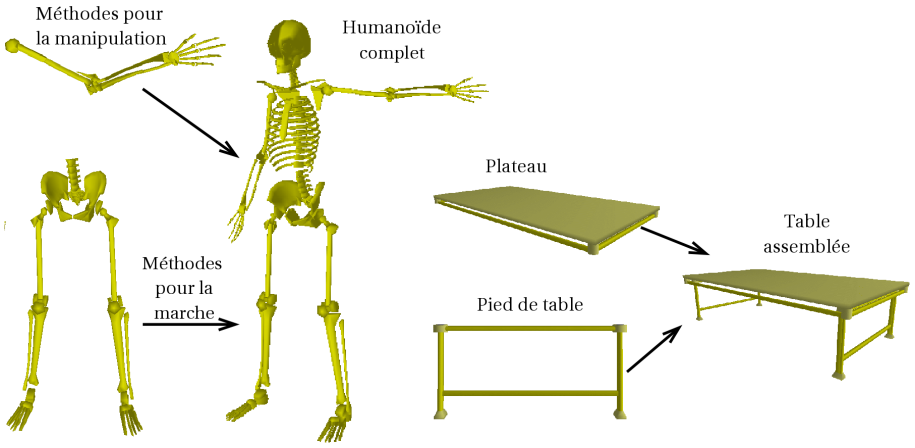


Figure 3.1: Exemple d'assemblage de chaînes cinématiques pour combiner des méthodes élémentaires ou pour créer des tâches d'assemblage.

Par exemple (figure 3.1), pour un robot humanoïde, nous pouvons envisager d'étudier séparément chacun des bras qui aura des méthodes locales propres, les jambes, la tête. Ce seront les *« chaînes cinématiques élémentaires »*. Bien entendu nous pouvons aussi étudier le robot complet en tant que *chaîne cinématique composée*. Celui-ci sera un assemblage des chaînes cinématiques élémentaires profitant de la combinaison des méthodes locales des sous-parties. De même pour une table que l'on envisage d'assembler nous pouvons considérer les pieds de la table, et son plateau comme des *« chaînes cinématiques élémentaires »*. Cela nous permettra d'aborder les tâches d'assemblage.

Comme le montre la figure 3.2, les *« chaînes cinématiques élémentaires »* commencent toujours par 6 degrés de liberté pour le placement dans le référentiel de l'environnement.

Cette propriété est importante, car ces chaînes cinématiques élémentaires vont être parfois attachées à d'autres robots et ainsi être positionnées en n'importe quel point de l'espace. Avec ce positionnement initial nous pouvons comparer les positions absolues des corps solides.

Notons que nous nous autoriserons certaines transformations sur ces degrés de liberté de placement afin de construire des roadmaps relatives (§ 4.3).

En d'autres termes, les chaînes cinématiques élémentaires représenteront les positions absolues des différents corps solides, elles seront utilisées pour la détection de collision.

Pour ces chaînes cinématiques nous pourrons définir un ensemble de méthodes locales propres. Par exemple pour un bras manipulateur nous pouvons avoir des mouvements dans l'espace des configurations (articulation du bras) ou dans l'espace cartésien de l'organe terminal[1] (figure 3.2.a). Pour une base mobile, nous avons toute une variété de mouvements possibles, allant de mouvements linéaires à un mouvement complexe tenant compte des contraintes de frottement d'une remorque (figure 3.2.b). Nous ne prendrons pas en compte les dynamiques dans nos exemples, mais les méthodes locales peuvent

[1]Un mouvement dans cette espace assurera un déplacement linéaire du repère de la pince entre deux configurations. Cela repose sur une méthode de cinématique inverse.

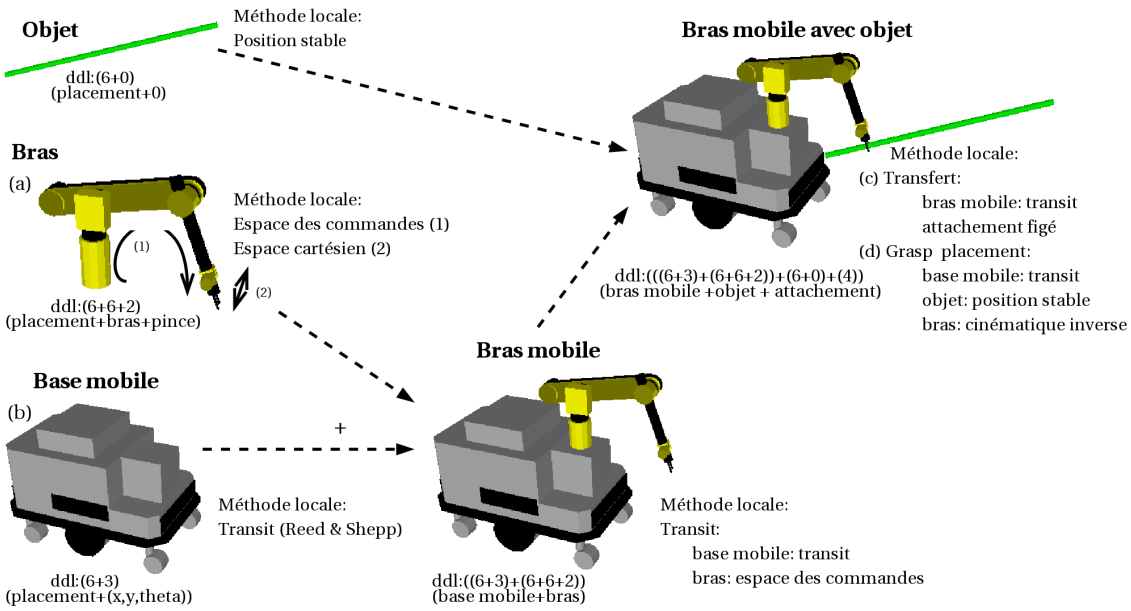


Figure 3.2: Exemple de combinaison de chaînes cinématiques et des concaténations des configurations correspondantes.

intégrer un certain nombre de contraintes propres à la dynamique des systèmes, assurant ainsi leur faisabilité[2]. Ces méthodes locales pourront être combinées pour créer les méthodes locales des *chaînes cinématiques composées*.

3.1.2 La composition de chaînes cinématiques.

Après avoir défini les chaînes cinématiques, nous pouvons définir une méthode de composition permettant d'assembler ces chaînes cinématiques en reliant entre elles leurs articulations. Ainsi dans l'exemple de la figure 3.2, le robot résultant a pour configuration une concaténation des configurations des chaînes cinématiques, auxquelles s'ajoutent les degrés de liberté des *chaînes cinématiques virtuelles* définissant les attachements.

En plus de cette concaténation, on peut définir pour le robot résultant un ensemble de méthodes de calcul sur la chaîne cinématique résultante. En effet, nous pouvons obtenir des chaînes cinématiques redondantes, c'est-à-dire qu'un certain nombre de degrés de liberté doivent être calculé en fonction des autres. Par exemple un robot qui transporte un objet, va imposer la position de l'objet en fonction de la position de son bras, autrement dit les six degrés de liberté de l'objet doivent être calculés en fonction de la position du bras (figure 3.2.c). D'un autre côté, si nous cherchons à déterminer la position du bras qui permet d'attraper un objet au sol, il nous faut calculer les valeurs des degrés de liberté du bras en fonction de la position au sol de l'objet (figure 3.2.d). Nous allons utiliser intensivement les méthodes de calcul de chaînes cinématiques développées par [Cortés 02, Han 00] (§ 2.5). Nous ne cherchons pas à avoir une présentation minimale des degrés de liberté des robots.

[2]La méthode Reeds et Shepp (succession de ligne et d'arc de cercle) impose un changement du rayon de courbure (i.e. de l'orientation des roues) instantané. D'autres méthodes locales peuvent être utilisées pour assurer un changement continue plus proche de la réalité.

Cette composition des chaînes cinématiques a plusieurs avantages :

- Une représentation unique des positions absolues des corps rigides par l'utilisation de *chaînes cinématique élémentaires*.
- Une relation simple entre les configurations des *chaînes cinématiques élémentaires* et les *chaînes cinématiques composées* permettant de relier entre eux des nœuds de différentes roadmaps.
  La configuration d'un robot portant un objet peut être séparée de façon aisée en une configuration du robot seul et une configuration de l'objet seul. Autrement dit, un nœud d'une roadmap de *Grasp* $\cap$ *Placement* (§ 2.7.2) peut-être lié facilement à un nœud d'une roadmap de *Transit* du robot seul et un nœud d'une roadmap des positions stables de l'objet.
- Une combinaison aisée des méthodes locales. On peut très bien avoir un robot manipulateur avec une base mobile qui suit une méthode locale de déplacement avec remorque, combinée avec une méthode locale de déplacement dans l'espace cartésien pour le bras. Bien entendu, il est tout à fait possible de créer une méthode locale nouvelle pour le nouveau robot composé.

La propriété que nous allons utiliser le plus souvent, est la relation entre configurations qui va nous permettre de définir les liens entre roadmaps. Elle sera expliquée plus en détail un peu plus loin dans ce chapitre § 3.3.

3.2 Les types de roadmaps.

Comme nous l'avons déjà mentionné, notre approche est basée sur la construction d'un certain nombre de roadmaps. Chacune d'entre elles ne se rapportera qu'à une seule chaîne cinématique (robot, objet, composition de robots et d'objet). Nous voulons nous assurer que les configurations des nœuds sont définies de manière unique. Nous allons voir ici les deux types de roadmaps que nous allons utiliser pour définir l'espace de recherche.

3.2.1 Les roadmaps de mouvements.

Elles servent à définir les mouvements valides des robots, autrement dit les méthodes locales utilisées pour construire ces roadmaps doivent décrire des mouvements faisables pour les robots. Les méthodes par échantillonnage (§ 2.2.1 et § 2.3.2) ou par diffusion (§ 2.4) peuvent être utilisées pour enrichir ces roadmaps.

Pour un problème de manipulation, elles peuvent représenter les mouvements de *Transit* d'un robot, les mouvements de *Transfert* d'un objet. Nous avons aussi choisi de les utiliser pour représenter les positions stables d'un objet.

Il faut noter ici la particularité des roadmaps de *Transfert*. Un chemin de *Transfert* valide doit avoir une position de prise fixe tout au long du mouvement. Ainsi deux nœuds d'une roadmap ne peuvent pas être liés entre eux s'ils correspondent à deux manières différentes de saisir l'objet. Nous allons ainsi avoir un ensemble de composantes connexes qui ne pourront pas être reliées directement entre elles.

Pour éviter l'explosion du nombre de composantes connexes, nous utiliserons plutôt une méthode par échantillonnage basée sur un sous-ensemble de configuration de prise (§ 4.2) ou une méthode de diffusion modifiée qui prend la même configuration de prise que le nœud le plus proche dans la roadmap.

Pour les objets, il peut sembler étrange de consacrer une roadmap à leurs positions stables. En effet, les objets ne pouvant se déplacer seuls, les nœuds seront forcément isolés. Mais ces nœuds seront utilisés à leur tour pour construire des positions de saisie pour les autres robots. Un objet qui est posé par un robot à un endroit peut-être pris par un autre robot par la suite. Cette position stable est accessible à tous les robots pour construire de nouveaux nœuds de saisi (§ 4.2).

La méthode locale d'une telle roadmap indiquera les mouvements possibles de l'objet, c'est à dire aucun mouvement. Elle échouera systématiquement pour relier deux nœuds distinctes. Comme nous l'avons dit précédemment, nous avons choisi de différencier les robots et les objets uniquement par leur méthode locale.

3.2.2 Les roadmaps affinables.

Les roadmaps affinables représentent tous les mouvements qui possèdent la propriété de réduction. Cela peut être aussi bien un mouvement linéaire qui peut être transformé en une suite de mouvements *Reeds et Shepp* [J.-P. Laumond 94] qu'un mouvement dans $Grasp \cap Placement$ qui peut être transformé en une succession de mouvements de *Transit* et de *Transfert* (§ 2.8.6).

Ces roadmaps sont très utiles, parce qu'elles permettent de remplacer la recherche d'un ensemble de chemins, par une méthode locale entre deux configurations. Une fois que l'on a trouvé une solution à travers ces roadmaps, il est alors possible d'en extraire une solution réalisable. Par contre l'extraction peut-être un peu compliquée et nécessiter une optimisation (§ 2.8.6).

Nous pouvons aussi remarquer, que dans le cas d'une implémentation sur un robot réel, il est aussi possible d'utiliser des méthodes locales qui ne possèdent pas la propriété de réduction mais possèdent une bonne propriété de convergence. Autrement dit il n'est pas possible de transformer un chemin $\mathcal{C}$ en une succession finie de mouvements faisables, mais il est toujours possible de trouver une suite convergente de mouvements faisables qui tend vers la position finale du chemin $\mathcal{C}$. Pour un robot réel, il ne sert à rien de viser une position absolument précise qu'il ne pourra de toute façon pas atteindre. Une solution approchée est tout à fait satisfaisante dans un tel cas.

3.3 Les liens entre roadmaps.

Comme nous l'avons sous-entendu précédemment il va y avoir un certain nombre de liens entre les roadmaps qui vont être basés sur les méthodes de composition des robots (§ 3.1.2).

Un robot va se déplacer dans la roadmap de *Transit* puis saisir un objet et ainsi passer dans la roadmap de $Grasp \cap Placement$. Il existe un lien entre ces roadmaps et plus précisément un lien entre le nœud de la roadmap de *Transit* qui permet de saisir l'objet et le nœud de la roadmap de $Grasp \cap Placement$ où la chaîne cinématique définit cet attachement.

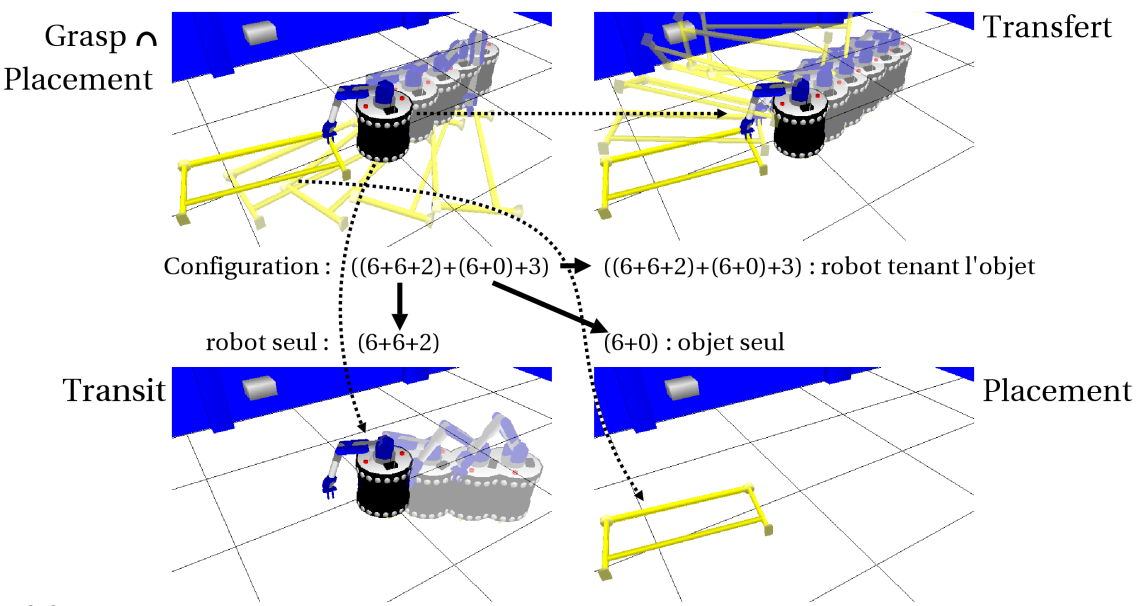


Figure 3.3: Exemple des liens d'héritages partant d'une roadmap de *Grasp* ∩ *Placement* vers les roadmaps de *Transfert*, *Transit* du robot et *Placement* de l'objet.

Nous appellerons ces liens entre nœuds de roadmap différentes des *« liens d'héritages »*.

Ces *« liens d'héritages »* permettent de reproduire fidèlement une partie de la configuration d'une roadmap dans une autre. Ils peuvent être utilisés entre n'importe quel type de roadmaps.

Ce lien commence forcément d'une roadmap basée sur un robot ayant une chaîne cinématique incluant toutes les chaînes cinématiques élémentaires de la roadmap de destination. Autrement dit, il part du robot le plus complexe vers les robots les plus simples.

Ainsi la roadmap de destination hérite d'une partie de la configuration de la roadmap source.

Par exemple (figure 3.3) un nœud de *Grasp* ∩ *Placement* peut conduire à un nœud de *Placement* pour l'objet et un de *Transit* pour le robot seul. Grâce aux définitions des compositions des chaînes cinématiques (§ 3.1.2), il est possible de retrouver quelles parties de la configuration d'un nœud de *Grasp* ∩ *Placement* doivent être reproduites dans les nœuds de *Placement* de l'objet ou de *Transit* du robot. Il en va de même pour le report d'un nœud de *Grasp* ∩ *Placement* dans une roadmap de *Transfert*.

Il faut noter que ce lien peut ne pas être systématique. Tous les nœuds de la roadmap de *Grasp* ∩ *Placement* ne seront pas obligatoirement liés à des *« sous-nœuds »*. Il faut choisir les nœuds suffisamment intéressants pour être transmis à une autre roadmap sans trop surcharger les roadmaps de destination. Nous verrons les méthodes d'enrichissement des roadmaps dans le chapitre suivant.

Un unique *« lien d'héritage »* définit une relation entre deux roadmaps, mais ne désigne pas forcément une action telle que la saisie d'un objet ou l'assemblage de plusieurs objets. Dans un monde fermé, il n'y a pas de disparition ou de création d'objets. Ainsi une action qui changerait la nature des objets (robot composé) fait aussi un changement de roadmaps. Mais ce changement n'est pas forcément entre une roadmap vers une autre mais peut être d'une vers plusieurs (pose d'objet, désassemblage)

ou de plusieurs vers une (prise d'objet, assemblage). Il existe par exemple une action entre *Grasp* ∩ *Placement* et le couple *Transit* du robot et *Placement* de l'objet. Nous représenterons ces actions par des *« arcs entre roadmaps »*.

Définition 6 *Nous définirons un* « arc entre roadmaps » *par un ensemble de* « liens d'héritages » *partant d'un même nœud d'une roadmap source* $\mathcal{R}_S$ *tel que l'ensemble des* « chaînes cinématiques élémentaires » *des roadmaps de destinations forment une partition des* « chaînes cinématiques élémentaires » *de* $\mathcal{R}_S$.

Ces *« arcs entre roadmaps »* permettront de définir des actions de composition tels que la saisie ou l'assemblage d'objets ou des changements de mode de déplacement tels que de *Grasp* ∩ *Placement* vers *Transfert*.

3.4 La topologie des roadmaps.

Tout au long de ce chapitre nous avons décrit l'utilisation de roadmaps pour capturer la topologie de l'espace libre des mouvements faisables par chaque robot (ou objet). Cette section a pour but de présenter un graphe appelé *« Graphe des Cinématiques Élémentaires »* (GCE) capable de lier les différentes roadmaps et d'exprimer aussi une topologie, mais cette fois-ci en prenant en compte tous les mouvements possibles. Ce qui va nous permettre de chercher une solution à un problème donné.

Nous illustrerons la définition de ce GCE à travers l'exemple de la figures 3.4. Celle-ci représente deux chariots élévateurs qui peuvent transporter une caisse plate. Il y a donc deux robots et un objet déplaçable. Il est de plus possible de passer l'objet directement d'un chariot à l'autre sans le poser sur le sol (figure 3.4 saisie simultanée).

3.4.1 La généralisation du graphe de manipulation.

Dans le chapitre précédent nous avons rappelé la notion de *« graphe de manipulation »* (§ 2.7.1) et notamment la version qui traite du problème des positions de prise et de pose continues. Dans ce cas les composantes connexes de *Grasp* ∩ *Placement* sont reliées entre elles par des arcs de *Transfert* et des arcs de *Transit*. Pour aider la recherche des arcs de *Transit*, une roadmap de *Transit* prenant en compte l'environement statique mais pas l'objet déplaçable était construite.

Vu sous un autre angle, ce graphe de manipulation reviendrait à relier les composantes connexes de *Grasp* ∩ *Placement* par des composantes connexes de *Transfert* ou des couples de composantes connexes (*Transit* du robot, *Placement* de l'objet). Pour chaque position de l'objet déplaçable, la composante de *Transit* est déterminée en ne prenant pas en compte les arcs qui entre en collision avec l'objet. Cela permet de limiter les tests de nouveaux arcs dans l'environnement statique.

Une des idées principales des *« Graphes des Cinématiques Élémentaires »* (GCEs) est de mettre en relation des composantes connexes des différentes roadmaps. Cette relation pourra se faire entre des ensembles de composantes connexes de différentes roadmaps pour représenter l'idée de composition

comme le couple (*Transit*, *Placement*). Elle représentera des actions changeant la nature des objets (changement de mode de déplacement entre *Grasp* ∩ *Placement* et *Transfert* ou composition de robots *Grasp* ∩ *Placement* et (*Transit*, *Placement*)).
⇒ Les GCEs vont mettre en relation les composantes connexes des différentes roadmaps.

Dans le cas de plusieurs robots ou plusieurs roadmaps, la roadmap de *Transit* doit être validée par rapport à tous les objets déplaçables et les robots de l'environnement. Il en va de même pour les roadmaps de *Grasp* ∩ *Placement* et de *Transfert* qui cette fois-ci doivent être aussi validées.

Nous avons vu que pour résoudre un problème avec plusieurs robots (§ 2.6.1) il est bien plus efficace dans un premier temps de planifier indépendamment pour chaque robot et de valider des chemins dans le produit cartésien des roadmaps individuelles.
⇒ Les GCEs vont aussi se servir de roadmaps qui devront être validées pour prendre en compte les objets déplaçables.

Le graphe de manipulation définit en tout point la configuration du robot et de l'objet. Sa généralisation directe devrait définir en tout point les configurations valides des robots et des objets. En fait, ce point ne sera pas généralisé.

En effet, s'il existe une configuration qui rassemble tous les robots, nous retrouvons la complexité de la planification multi-robots (§ 2.6.1). De plus dans ce cas chaque robot pourra avoir différents types de mouvements.

Par exemple, une configuration d'un robot R_1 en *Transit* peut être combinée avec : une configuration d'un robot $R_2 - O2$ transportant un objet (*Transfert* ou *Grasp* ∩ *Placement*) ou avec le robot R_2 en *Transit* et l'objet O_2 en *Placement*. Il y a déjà trois méthodes définies avec R_2 et O_2 qu'il faut combiner avec R_1. Cela devient beaucoup plus complexe quand le nombre de robots augmente. Là où un chemin de *Transit* d'un robot R_1 devait être validé par rapport à une position de l'objet déplaçable, il faut maintenant le coordonner avec tous les mouvements possibles faisables par les autres robots et objets déplaçables de l'environnement.

Cette idée sera le second point fort des GCEs. Il sera possible de représenter cette notion : un robot se déplace seul et les autres objets agissent d'un autre coté. La coordination ne sera pas calculée pour éviter la complexité combinatoire. Il faudra donc une phase de validation pour vérifier l'existence d'un chemin coordonné.
⇒ Les GCEs sont construits hiérarchiquement afin de résoudre indépendamment des sous-parties d'un problème global et de combiner les solutions entre elles.

Il existera un GCE pour chaque sous ensemble de robots et d'objets qui pourront se mouvoir indépendamment du reste.

Le problème peut être divisé en sous-problèmes : l'objet seul, le robot seul, le robot et un objet, les deux robots, ... Par exemple dans le cas de l'objet seul, 7 nœuds isolés ont été tirés aléatoirement. Ils représentent 7 placements possibles. Le GCE de l'objet seul est donc représenté par 7 nœuds isolés du GCE donc 7 composantes connexes (figure 3.5.a).

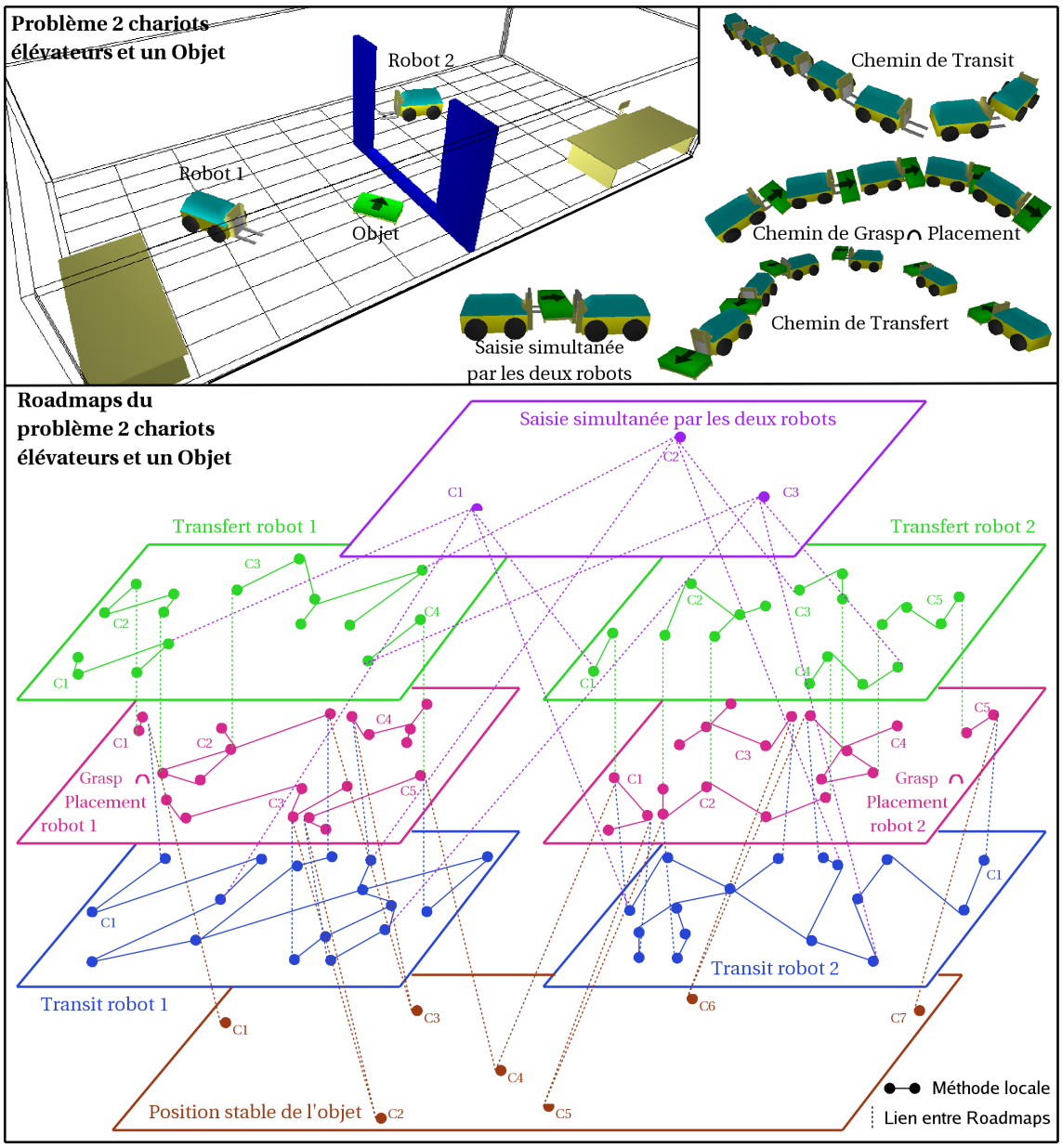


Figure 3.4: Topologie d'un problème de manipulation de deux chariots élévateur et d'un objet.

Pour le robot seul il n'y a qu'une seule composante connexe. Les relations entre roadmaps se font en fait au niveau des composantes connexes. C'est ainsi que nous avons décidé qu'un nœud d'un GCE représente une composante connexe. Ainsi le GCE du robot seul (figure 3.5.b) a un nœud et donc une seule composante connexe.

Pour le robot avec l'objet (figure 3.5.c) il est possible de déplacer le robot en *Transit* avec l'objet dans une des 7 positions possibles des roadmaps. Cela est représenté par la composition de la composante connexe de la roadmap de *Transit* du robot seul et du *Placement* de l'objet. Il y a donc 7 compositions possibles. De telles compositions sont aussi des nœuds du GCE.

Ces nœuds sont reliés à *Grasp* ∩ *Placement* à travers des arcs entre roadmaps (§ 3.3) qui représentent des opérations de prise et de pose. Ces arcs entre roadmap représentent les liens entre les nœuds d'un GCE.

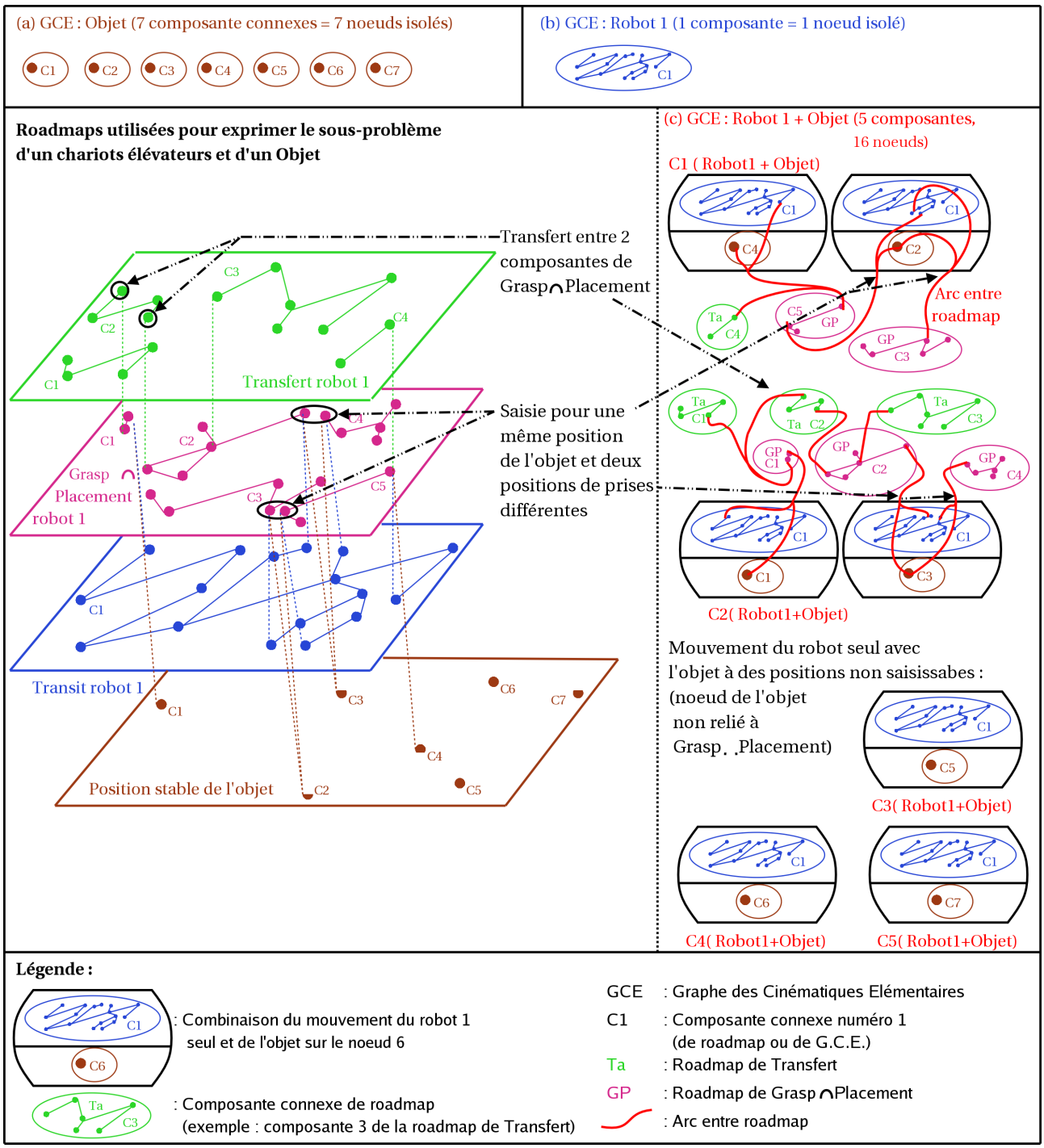


Figure 3.5: Graphe des Cinématiques Élémentaires pour le sous-problème d'un robot et d'un objet. (a) sous-GCE pour l'objet seul, (b) sous-GCE pour `robot1` seul, (c) GCE pour l'ensemble `{Robot1, Objet}`.

La figure 3.5 montre les 5 composantes connexes du GCE représentant le premier chariot élévateur et l'objet. Elles ont été construites en prenant en compte les 7 positions de l'objet. Pour le problème restreint à ces deux éléments il est impossible de passer d'une composante connexe à l'autre. Cela permet de représenter tous les mouvements possibles du sous-problème.

3.4.2 Définition des « Graphes des Cinématiques Élémentaires ».

Nous avons élaboré les *« Graphes des Cinématiques Élémentaires »* afin de représenter les mouvements possibles dans le cadre d'un sous-problème du problème à traiter et afin de pouvoir les assembler de manière hiérarchique.

Un *« Graphe des Cinématiques Élémentaires »* (GCE) $\mathcal{G}$ est donc relatif à un sous-ensemble $\mathcal{E}$ des chaînes cinématiques élémentaires.

Il peut y avoir deux types de nœuds : les *« nœuds de roadmap »* et les *« nœuds de composition »*. Ces nœuds peuvent être reliés par deux types d'arcs : les *« arcs entre roadmaps »* et les *« arcs d'équivalence »*.

Les composantes connexes ainsi crées représenteront l'ensemble des actions faisables par l'ensemble des robots et des objets représentables par l'ensemble $\mathcal{E}$ des chaînes cinématiques élémentaires du GCE.

3.4.2.1 Les « nœuds de roadmap ».

Les *« nœuds de roadmap »* correspondent à une composante connexe d'une roadmap de mouvement ou d'une roadmap affinable ayant le même ensemble $\mathcal{E}$ de chaînes cinématiques élémentaires que le GCE (figure 3.6.a).

Ils permettent de représenter les mouvements des robots et des objets isolés ou composés, par exemple des mouvements de *Transit*, de *Grasp* $\cap$ *Placement* de *Transfert* ou même les *Placement* de l'objet.

Ce qui est pris en compte, c'est l'ensemble des chaînes cinématiques élémentaires et donc l'ensemble des corps rigides du problème. Par contre, les chaînes cinématiques virtuelles ne sont pas prises en compte. Ainsi deux robots composés ayant les mêmes chaînes cinématiques élémentaires mais ayant des définitions d'attachement différentes seront présents dans le même GCE.

3.4.2.2 Les « nœuds de composition ».

Définition 7 *Le GCE $\mathcal{G}_i$ est un sous-GCE de $\mathcal{G}_j$ si et seulement si l'ensemble des chaînes cinématiques élémentaires $\mathcal{E}_i$ de $\mathcal{G}_i$ est inclue dans l'ensemble des chaînes cinématiques élémentaires $\mathcal{E}_j$ de $\mathcal{G}_j$.*

Définition 8 *L'ensemble des sous-GCEs $\mathcal{G}_i$ du GCE $\mathcal{G}$ forment une partition de $\mathcal{G}$ si et seulement si les ensembles de chaînes cinématiques élémentaires $\mathcal{E}_i$ associées à $\mathcal{G}_i$ forment une partition de l'ensemble des chaînes cinématiques élémentaires $\mathcal{E}$ de $\mathcal{G}$.*

Les *« nœuds de composition »* du GCE $\mathcal{G}$, correspondant à un ensemble de composantes connexes de sous-GCEs $\mathcal{G}_i$ de $\mathcal{G}$ tels que les $\mathcal{G}_i$ forment une partition de $\mathcal{G}$ (figure 3.6.b).

Cela revient à dire qu'il est possible de combiner les solutions trouvées pour une partie des chaînes cinématiques élémentaires avec celles trouvées pour une autre partie. Par exemple le robot en *Transit* et l'objet en position fixée.

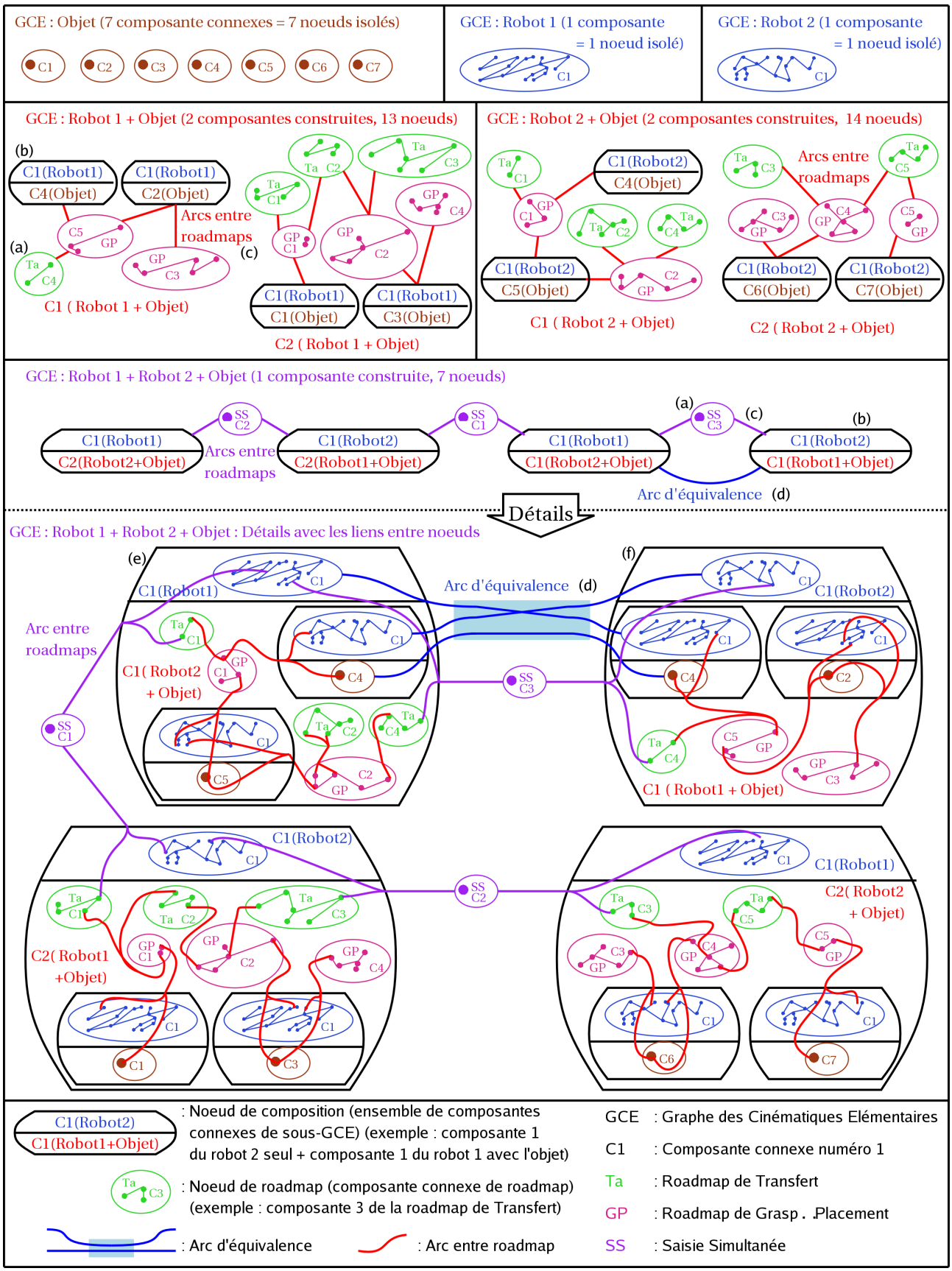


Figure 3.6: Topologie d'un problème du manipulation de deux chariots élévateur et d'un objet avec les Graphes des Cinématiques Élémentaires.

Mais ceci peut représenter aussi des mouvements beaucoup plus complexes de la part des robots élémentaires. La figure 3.6 montre notamment avec le détail du GCE du problème complet, que la combinaison d'un robot se déplaçant seul et du GCE du second robot et de l'objet peut être complexe.

Toutefois pour éviter une trop grande complexité nous avons décidé que les GCEs ne prennent pas en compte les interactions entre les différents sous-problèmes.

Ce type de nœud permet de réutiliser les roadmaps dans différents contextes et de ne pas construire le graphe du produit des configurations, ce qui correspondrait à faire le graphe produit des nœuds des composantes connexes des GCEs $\mathcal{G}_i$. La complexité d'un GCE est donc bien moins importante que celle du produit cartésien des nœuds des roadmaps.

Dans notre implémentation, nous ne construisons les *« nœuds de composition »* que s'ils sont reliés à une roadmap qui comprend toutes les chaînes cinématiques du GCE. Ainsi les nœuds des composantes 3, 4 et 5 de la figure 3.5.c ne sont pas construits (sauf s'ils appartiennent à l'état initial ou final du problème). Puisque ces nœuds composés ne sont pas atteignables, il est inutile de surcharger le *« Graphe des Cinématiques Élémentaires »*.

De même pour réduire encore le nombre de ces nœuds, nous ne représentons pas les compositions qui seraient déjà présents dans un autre nœud. Par exemple les nœuds (e) et (f) du GCE du problème complet contiennent les composantes connexes (respectivement 1, 1 et 4) des GCEs du robot 1 seul, du robot 2 seul et de l'objet seul. Il est donc inutile de créer un *« nœud de composition »* supplémentaire pour représenter cette composition.

3.4.3 Les « arcs entre roadmaps ».

Les *« arcs entre roadmaps »* sont issus de la définition des liens d'héritage entre roadmaps (§ 3.3) et de la loi de Lavoisier : conservation de la matière donc conservation des chaînes cinématiques élémentaires.

Ainsi, un nœud d'une roadmap de $Grasp \cap Placement$ qui possède des liens héritage vers les roadmaps de *Transit* et de *Placement* d'un objet, permet de définir un arc entre roadmaps entre une composante connexe de $Grasp \cap Placement$ et la combinaison des composantes connexes de placement de l'objet et de *Transit* du robot (figure 3.6.c).

3.4.4 Les « arcs d'équivalence ».

Les arcs d'équivalence sont définis pour faire apparaître l'utilisation commune dans deux nœuds de la même combinaison de composantes connexes de sous-GCEs $\mathcal{G}_i$ (figure 3.6.d, e et f).

Dans l'exemple de la figure 3.6, cela correspond au fait de poser l'objet au sol sur le nœud 4 avant de reprendre par un autre robot. Par définition, ce lien ne peut apparaître qu'entre deux *« nœuds de composition »*.

Ces arcs n'apparaissent qu'à partir d'une certaine complexité du problème. Ils ne représentent aucune action, seulement l'équivalence d'une partie de deux représentations différentes (figure 3.6.e et f).

3.4.5 L'utilisation des « Graphes des Cinématiques Élémentaires ».

Dans le cas de la manipulation un objet et un robot (figure 3.5) on peut remarquer l'équivalence de cette représentation avec le graphe de manipulation développé par Anis Sahbani (§ 2.7.3). Les composantes connexes de *Grasp* ∩ *Placement* sont reliées par des chemins de *Transfert*, ou des chemins de *Transit* avec la position de l'objet fixé. La seul différence c'est que les chemins de *Transit* restent encore à être validés par rapport à l'objet.

Dans le cas de plusieurs robots, les GCEs conservent une propriété de connexité tout en y ajoutant une notion de hiérarchie. Un ensemble de robots et d'objets avec des positions données sera représenté par un ou plusieurs nœuds de ce graphe. Si les positions initiales et les positions finales des robots et des objets se retrouvent dans une même composante connexe du GCE de plus haut niveau (impliquant l'ensemble des chaînes cinématiques du problème) alors il existe un chemin à travers l'ensemble des roadmaps permettant d'aller de l'état initial à l'état final dans le problème.

De plus si un problème peut être réalisé par un sous-ensemble de robots, alors les nœuds initiaux et finales du GCE global seront confondus. La représentation hiérarchique nous permet de voir à partir de quel moment ces nœuds sont distincts, et donc quel est le sous-ensemble minimum d'éléments nécessaires à la résolution du problème.

La figure 3.7 montre la recherche des passages obligatoires pour la résolution de deux problèmes. Dans le premier cas, les états initial et final sont confondus au niveau du GCE des 2 robots et de l'objet. Le problème peut se résoudre uniquement dans la composante 2 du GCE du robot 2 et de l'objet. Le robot 2 est obligé de saisir l'objet en $N7$ puis de passer par les composantes $C5$ de *Grasp* ∩ *Placement*, $C5$ de *Transfert*, $C4$ de *Grasp* ∩ *Placement* pour pouvoir le reposer en positon finale.

Dans le second cas, il est nécessaire de faire intervenir les deux robots. Il faut noter qu'il y a plusieurs chemins possibles. Il est possible de poser l'objet en position finale soit par le robot 1 (figure 3.7.b) soit par le robot 2 (figure 3.7.a). Par contre il est obligatoire de passer par les deux points de saisie simultanée $C1$ et $C2$.

Le choix des chemins possibles se fera au moment de la validation des coordinations des *« nœuds composés »*.

Les GCEs généralisent le graphe de manipulation au problème de plusieurs robots et de plusieurs objets en ayant une représentation compacte et hiérarchique du problème.

Toutefois cette représentation ne calcule pas les interactions entre les mouvements pouvant s'effectuer en parallèle. Cela a pour conséquence de réduire la complexité de l'espace de recherche, mais aussi de nécessiter l'utilisation d'une étape de validation.

Cette étape de validation pourra utiliser l'information de connexité pour connaître l'ensemble des actions nécessaires pour atteindre le but. La connexité des GCEs est une condition nécessaire, mais pas suffisante.

Il nous reste bien sûr à voir dans la section suivante comment nous allons combiner toutes les techniques développées dans les sections précédentes pour choisir quelles roadmaps et quelles méthodes utiliser pour enrichir les GCEs.

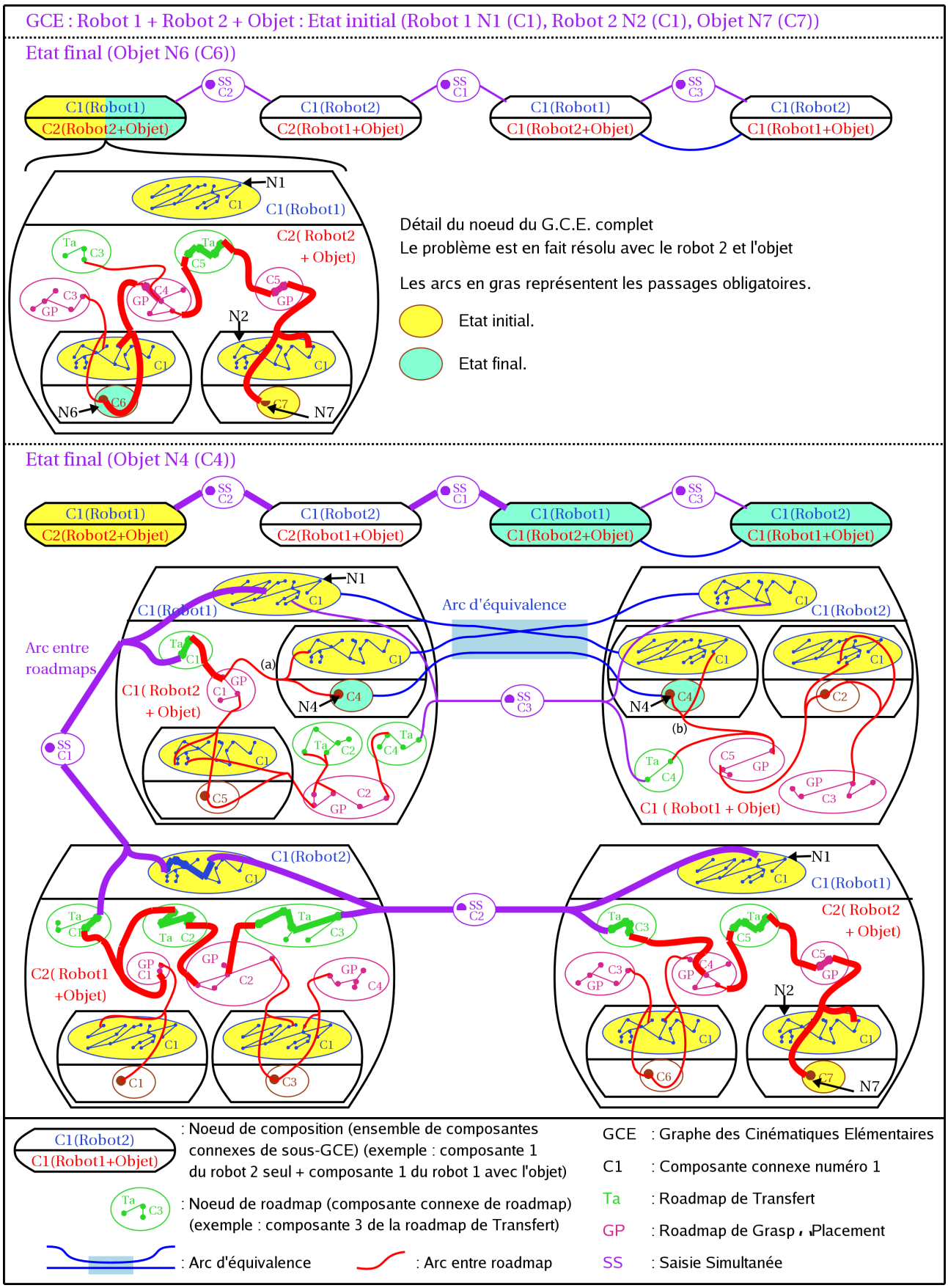


Figure 3.7: Exploitation de la topologie pour connaître les passages obligatoires à la résolution d'un problème.

3.5 Conclusion.

Nous avons développé ici un formalisme qui permet de représenter les problèmes de manipulations traités aujourd'hui dans la littérature, tout en se généralisant à des problèmes avec plusieurs robots et plusieurs objets.

Il nous reste bien sûr à voir dans le chapitre suivant comment étendre les roadmaps et donc enrichir les GCEs.

De plus, comme nous avons choisi de ne pas explorer l'espace produit cartésien des espaces de chaque robot à cause de l'explosion combinatoire exponentielle en nombre de robot, il va falloir une étape de validation des chemins trouvés dans les GCEs qui prenne en compte la présence des autres robots et objets.

Cette validation ne va pas pouvoir se faire de manière hiérarchique et par connexité (même si cela reste un bon guide), mais il faudra utiliser une validation pour chaque mouvement.

Notre premier test fut d'utiliser un algorithme du type A* pour rechercher une solution. Malheureusement l'espace de recherche est trop grand pour une approche directe. Les GCEs gardent le parallélisme des plans et l'ensemble des nœuds possibles. Il nous a donc fallu imaginer une nouvelle approche qui permette de faire des choix dans les roadmaps à étudier et qui regroupe les nœuds en classe d'équivalence. Cette nouvelle approche est basée sur un état qui représente des classes d'équivalence des nœuds et sur lequel on peut appliquer un ensemble d'actions permettant de choisir quels arcs du GCE traverser.

C'est ainsi qu'est apparu notre première représentation symbolique du problème. Nous verrons dans le chapitre 5 comment nous allons regrouper les nœuds des roadmaps en position symbolique et comment nous allons pouvoir utiliser les techniques issues de la planification de tâches pour rechercher des solutions à travers les GCEs De plus nous allons pouvoir traiter des problèmes nouveaux qui intègrent à la fois les problèmes de manipulation géométriques et des tâches plus symboliques.

Chapitre 4

L'apprentissage de la topologie.

Après avoir défini une nouvelle représentation de l'espace de recherche à l'aide des *« Graphes des cinématiques Élémentaires »* dans le chapitre précédent, il nous faut maintenant décrire comment explorer cet espace, donc comment enrichir les roadmaps.

La planification de mouvement possède un ensemble d'outils performants pour résoudre des problèmes de déplacement ou de manipulation. Un point crucial qui est apparu dans l'état de l'art du chapitre 2 est le fait qu'il est important de restreindre l'espace dans lequel sont tirés les nœuds des roadmaps.

Que ce soit pour la détection des passages étroits (§ 2.3.1), les chaînes cinématiques fermées (§ 2.5.2), ou la manipulation (§ 2.8.5), les algorithmes sont plus performant quand l'espace de recherche est restreint.

Ce chapitre à pour vocation d'expliquer où et comment vont être enrichies les roadmaps.

Pour cela nous allons définir de nouvelles méthodes qui vont permettre le limiter la recherche : les *« roadmaps heuristiques »* (§ 4.1), l'enrichissement des roadmaps par recomposition (§ 4.2), les *« roadmaps relatives »* (§ 4.3).

Puis nous finirons par la présentation d'un moteur de systèmes de règles heuristiques qui régie le choix des roadmaps et des méthodes à utiliser pour apprendre la topologie de l'environnement (§ 4.4).

4.1 Les roadmaps heuristiques.

Dans le chapitre précédent, nous avons montré l'utilisation des *« roadmaps de mouvements »* et des *« roadmaps affinables »*.

Les roadmaps heuristiques ne représentent pas des mouvements faisables par les robots, mais sont très utiles pour construire les autres roadmaps. Elles sont utilisées principalement pour deux raisons

différentes :

- Pour explorer la topologie d'un problème relaxé.
- Pour limiter une exploration qui pourrait rendre la recherche trop complexe.

Le meilleur exemple de limitation d'exploration apparaît avec une roadmap de *Transfert* qui a une composante connexe par position de prise.

Pour avoir une connexité intéressante, il peut être utile de limiter la progression des configurations de prise par rapport à l'extension des roadmaps.

Pour cela les configurations de prise vont être stockées sous forme de nœuds, dans une roadmap heuristique. Il suffit alors que cette roadmap heuristique croisse faiblement par rapport à la roadmap de *Transfert* pour que l'on puisse « puiser » des configurations de prises pour trouver des chemins de transfert utiles. C'est une manière de restreindre l'espace de recherche pour étendre la connexité avant de chercher de nouvelles prises.

La recherche de la topologie d'un problème relaxé est une des propriétés les plus intéressantes des roadmaps heuristiques.

Prenons l'exemple de la manipulation avec des prises et des poses continues. Il faut trouver les positions intermédiaires intéressantes. Pour cela il est possible de comparer la topologie de $Grasp \cap Placement$ avec la topologie de l'objet s'il était capable de bouger par lui-même.

Si l'on construit par exemple, une roadmap heuristique dans lequel l'objet peut se déplacer seul dans *Placement* (figure 4.1), un mouvement dans $Grasp \cap Placement$ ne sera faisable que s'il existe un mouvement pour l'objet seul. Ainsi si nous n'avons pas de connexité dans cette roadmap heuristique, il peut être nécessaire de rechercher un chemin de *Transfert* pour relier les composantes connexes de $Grasp \cap Placement$.

Si, d'un autre côté, nous avons un chemin faisable pour l'objet seul, mais infaisable dans $Grasp \cap Placement$, nous pouvons peut-être détecter une position de *« re-grasping »* (§ 2.8.5), c'est-à-dire une position unique de l'objet qui est relié à deux composantes connexes de $Grasp \cap Placement$ non reliées entre elles, mais qui peuvent alors être liées par un chemin de *Transit*.

La topologie de cette roadmap permet donc de savoir dans quelle roadmap chercher des chemins solution.

Une autre utilisation possible pour cette recherche de topologie est de trouver un chemin faisable pour une sphère englobante et utiliser les nœuds de cette roadmap grossière pour trouver un chemin pour un robot plus complexe. Mais nous avons peu étudié cet aspect pendant cette thèse.

Pour ce cas particulier, il n'est toutefois pas possible d'utiliser les *« liens d'héritage »* entre roadmap (§ 3.3) qui supposent des chaînes cinématiques au moins partiellement identiques.

Nous avons donc développé les *« liens heuristiques »*. Ils ressemblent aux *« liens d'héritage »* dans le sens qu'ils définissent une copie d'une partie de la configuration d'un nœud source vers une partie de la configuration d'un nœud destination. En revanche, il n'y a jamais création systématique de sous-nœuds. Ils ne conservent aucune mémoire des relations entre nœuds. Il peut aussi n'y avoir aucune relation entre les chaînes cinématiques des deux roadmaps.

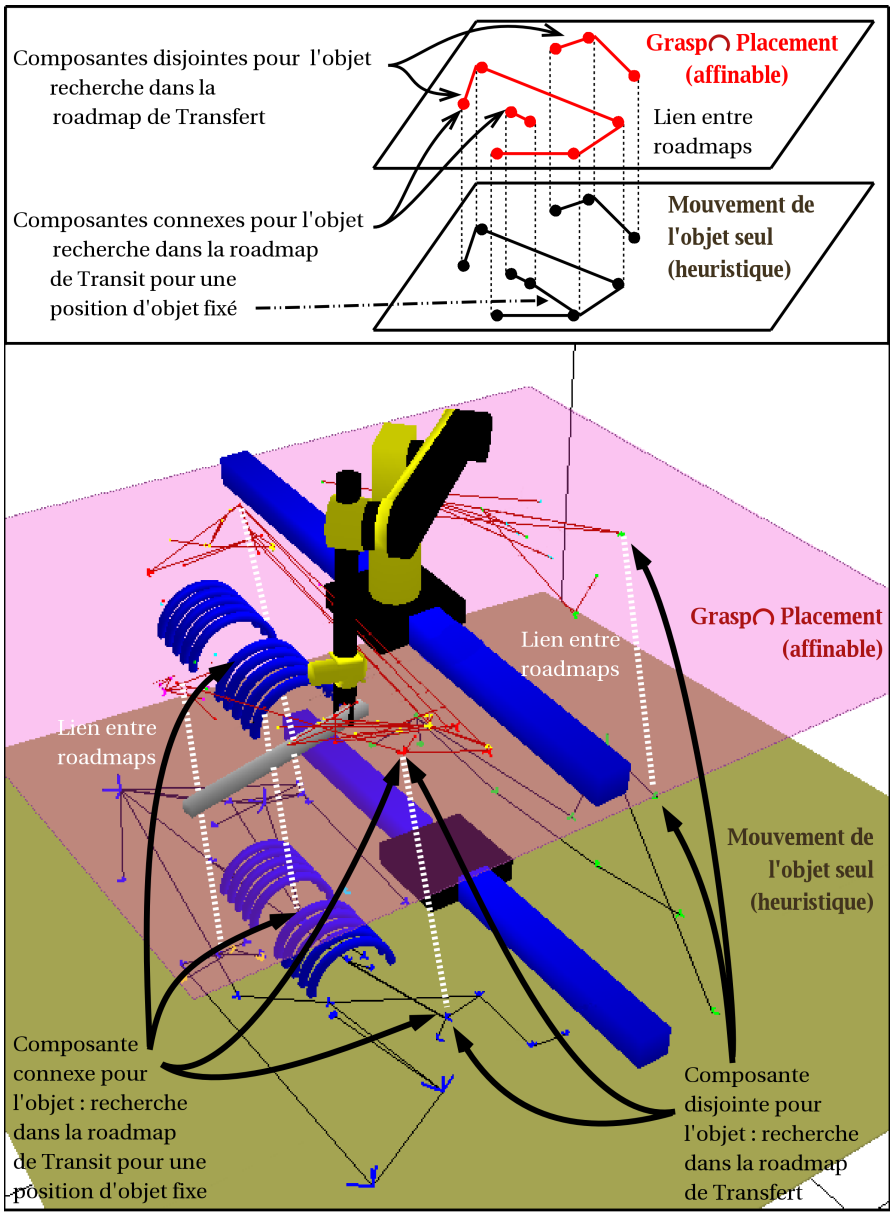


Figure 4.1: Utilisation d'une roadmap heuristique pour étudier la topologie d'une roadmap de *Grasp* ∩ *Placement*.

Il n'y a alors aucune garantie qu'un nœud source valide conduise à la création d'un nœud destination valide. Par contre, de la même manière que pour les liens d'héritage, il peut être utilisé pour construire de nouveaux nœuds dans d'autres roadmaps par la méthode présentée ci-dessous.

4.2 Enrichissement des roadmaps par recomposition.

Nous avons vu qu'il était possible à partir d'une roadmap complexe de diviser la configuration d'un nœud pour créer des *« sous-nœuds »* dans les roadmaps de destination des liens d'héritages (§ 3.3). Ainsi un nœud dans *Grasp* ∩ *Placement* peut servir à la création d'un nœud de *Transit* du robot et de *Placement* de l'objet. Mais il peut aussi servir pour créer des nœuds de configuration de prise d'une roadmap heuristique.

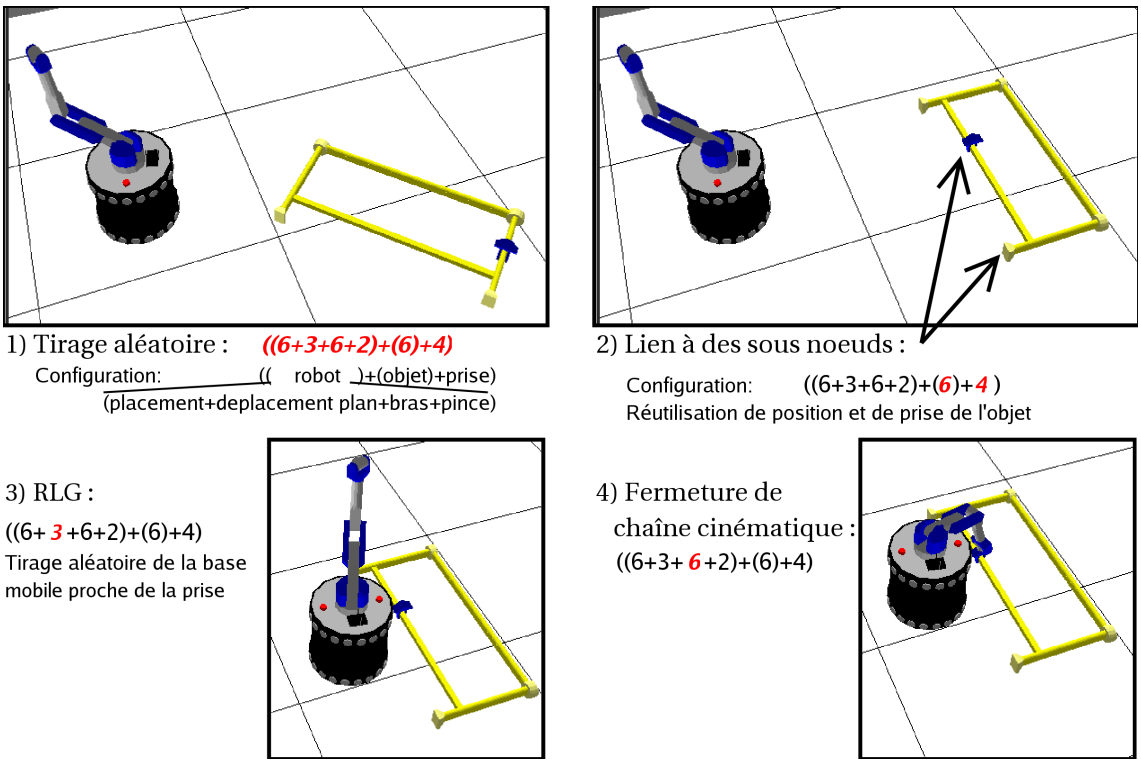


Figure 4.2: Les 4 étapes de la création d'un nœud.

De manière opposée, nous pouvons nous servir de « *sous-nœuds* » pour créer au moins une partie de la configuration d'un nœud à ajouter dans une autre roadmap. Au lieu de partir d'une configuration totalement aléatoire, une partie proviendra de « *liens d'héritage* » ou de « *liens heuristiques* ».

Bien entendu il est possible d'utiliser plusieurs liens pour construire une configuration plus complexe. Par exemple nous pouvons utiliser un nœud correspondant à la position d'un objet et un nœud correspondant à une configuration de prise pour essayer de générer un nœud dans $Grasp \cap Placement$ avec une position d'objet et une configuration de prise données (figure 4.2).

Nous avons combiné cette méthode avec le « *random loop generator* » (RLG § 2.5.2) pour restreindre l'espace de tirage aléatoire de certains degrés de liberté, par exemple pour avoir de bonnes chances de fermer la chaîne cinématique en positionnant la base du robot dans un rayon acceptable de l'objet.

Pour résumer, la création d'un nouveau nœud se fait en 4 étapes (figure 4.2) :

1. Tirage aléatoire d'une configuration
2. Application des liens sur les sous-nœuds. C'est-à-dire copie des parties de configurations concernées (et application des changements de repères pour les liens relatifs).
3. Utilisation du RLG pour les cas de chaînes fermées. Tirage aléatoire de degrés de liberté limités par la valeur d'autres degrés de liberté déjà tirés afin d'augmenter la probabilité de fermeture de chaînes cinématiques.
4. Application des contraintes et calcul des degrés de liberté passifs (fermeture de chaîne cinématique). Si les contraintes ne sont pas satisfaites ou que le nœud est en collision on recommence à partir de l'étape 1 (un certain nombre de fois).

Grâce à cette méthode il est possible d'utiliser un nœud de *Placement* de l'objet pour pouvoir créer un nœud de saisie dans $Grasp \cap Placement$. Cela peut être utile pour changer de prise pour un robot, ou pour passer l'objet à un autre robot en le posant tout d'abord au sol.

Toutes les contraintes de la manipulation (prise fixée pendant le transfert, position de l'objet fixé s'il n'est pas saisi) et les contraintes de fermetures de chaînes cinématiques peuvent être respectées grâce à l'utilisation de cette méthode de tirage aléatoire des nœuds.

4.3 Les roadmaps relatives.

4.3.1 Intérêt des roadmaps relatives.

L'utilisation de roadmaps ne prenant pas en compte explicitement les autres robots et objets pose un problème pour la saisie. La méthode précédente montre comment obtenir une position de prise mais pas comment l'atteindre. Or c'est un mouvement délicat car proche de l'objet à saisir. Il faut utiliser une méthode de diffusion (§ 2.4) qui prend en compte l'objet à saisir pour pouvoir y arriver.

Même dans ce cas, la recherche d'un chemin qui s'approche du contact est coûteuse. Il faudrait capitaliser la recherche effectuée. Les roadmaps relatives tentent de répondre à ces deux attentes : une méthode spécifique qui prend explicitement en compte l'objet à saisir, une possibilité de réutiliser les résultats pour différentes opérations de saisie.

Pour cela elles possèdent des fonctions de changement de repères qui vont permettre de la définir relativement à un objet. Il sera possible de réutiliser ces roadmaps pour d'autre positions de l'objet.

Nous pouvons rapprocher l'utilisation de ces roadmaps relatives des *« kinematic roadmaps »* de LaValle [LaValle 99b] (§ 2.5.3). Dans le cas des *« kinematic roadmaps »*, pour éviter les problèmes dûs aux auto-collisions, une roadmap de mouvements du robot seul est construite ce qui permet de connaître les chemins faisables pour les chaînes cinématique fermées en évitant les auto-collisions. Cette roadmap peut-être alors reportée en chaque nœud d'une autre roadmap représentant les positions absolues. La connexité au sein de la *« kinematic roadmap »* définit alors les chemins qui sont valides du point de vu des auto-collisions dans la roadmap des positions absolues du robot. Il ne reste plus qu'à vérifier les collisions avec l'environnement.

Pour notre part nous utilisons principalement ces roadmaps afin de connaître les mouvements d'approche pour la saisie d'objets. Cette fois-ci ce sont les tests de collisions entre l'objet à saisir et le robot qui sont primordiaux. Nous utilisons généralement pour ces roadmaps des méthodes de diffusion en partant de la position de saisie afin de s'éloigner de celle-ci.

Nous utilisons aussi deux distances, l'une pour donner le diamètre minimal en dessous duquel nous sommes trop proches de l'objet pour pouvoir considérer que le robot est dégagé, la seconde pour donner le diamètre maximal au-delà duquel les points sont trop éloignés pour être « intéressants ».

L'utilisation de ces deux distances permettent aussi de limiter le problème d'effet de bord dont peuvent souffrir certains algorithmes de diffusion. Entre ces deux distances nous aurons ce que nous appelons des *« nœuds d'échappement »* (figure 4.3).

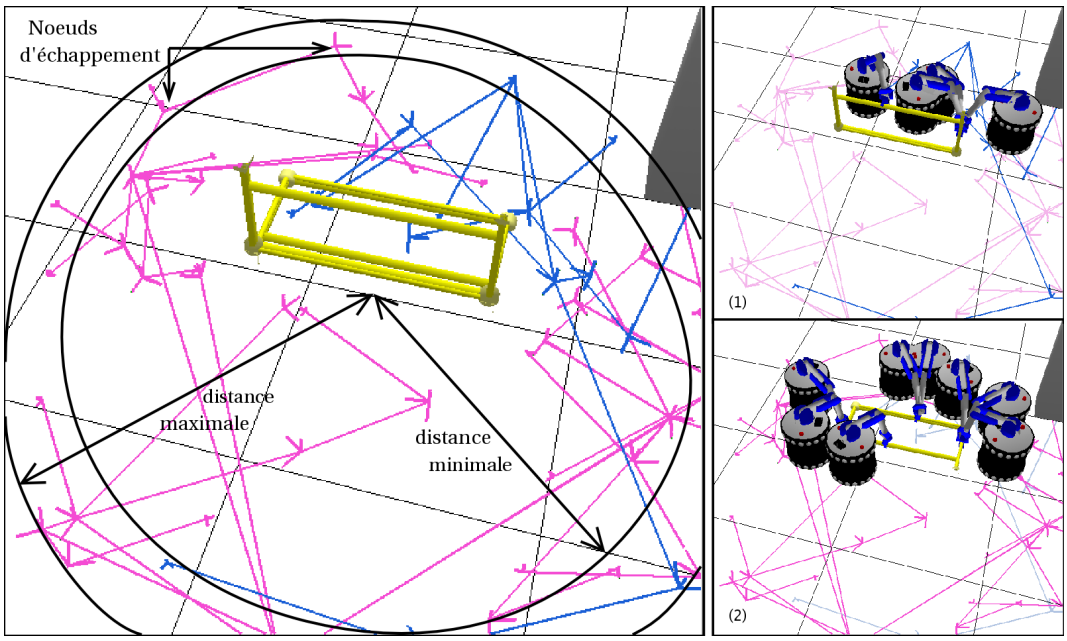


Figure 4.3: Exemple d'une roadmap relative : Méthode de diffusion avec distance aux objets pour la création de nœuds d'échappement. (1) et (2) sont deux composantes connexes qui ne peuvent pas être reliées car relatives à des objets de configurations différentes après changement de repère (transformation plan (x, y, θ)).

Remarque :

Nous retrouvons ici, une propriété des roadmaps de *Transfert* : la possibilité d'avoir des composantes connexes qui ne peuvent pas être reliées. En effet, les configurations du robot sont définies de manière relative à l'objet qui est dans une configuration donnée.

Par exemple si nous utilisons une transformation relative plane, c'est-à-dire que l'on doit faire uniquement une transformation plane (translation en x, y et rotation autour de z) pour passer de la roadmap relative à la roadmap absolue, alors une différence de hauteur de l'objet définira différentes composantes connexes. Il paraît normal qu'un robot doive se pencher pour saisir un objet au sol et se lever pour le saisir en haut d'une étagère. Il n'est pas possible de réutiliser le chemin trouver pour l'objet au sol pour saisir l'objet en haut de l'étagère.

Que ce soit les « *kinematic roadmaps* » de LaValle (§ 2.5.3) ou les roadmaps relatives, l'intérêt de ces méthodes est de pouvoir utiliser des techniques d'échantillonnage probabiliste différentes quand nous testons des collisions pour des objets proches, et de réutiliser les résultats obtenus.

En effet dans un grand environnement, les méthodes par échantillonnages par visibilité (§ 2.3.2) permettent d'avoir peu de nœuds ce qui est pratique pour la réutilisation des roadmaps. Par contre elles ont des difficultés à trouver des chemins pour la saisie d'objet. D'un autre coté, les méthodes par diffusions sont souvent assez lourdes en nombre de nœuds. L'utilisation des roadmaps relatives permettent donc de faciliter la combinaison de ces deux approches pour être efficace et réutilisable.

4.3.2 L'utilisation des roadmaps relatives.

L'utilisation des roadmaps relatives met en jeu trois roadmaps. Il y a la roadmap de *Grasp* ∩ *Placement* qui permet de définir la position de saisie de l'objet, la roadmap relative et la roadmap de

Transit du robot seul qui représentera les positions absolues du robot.

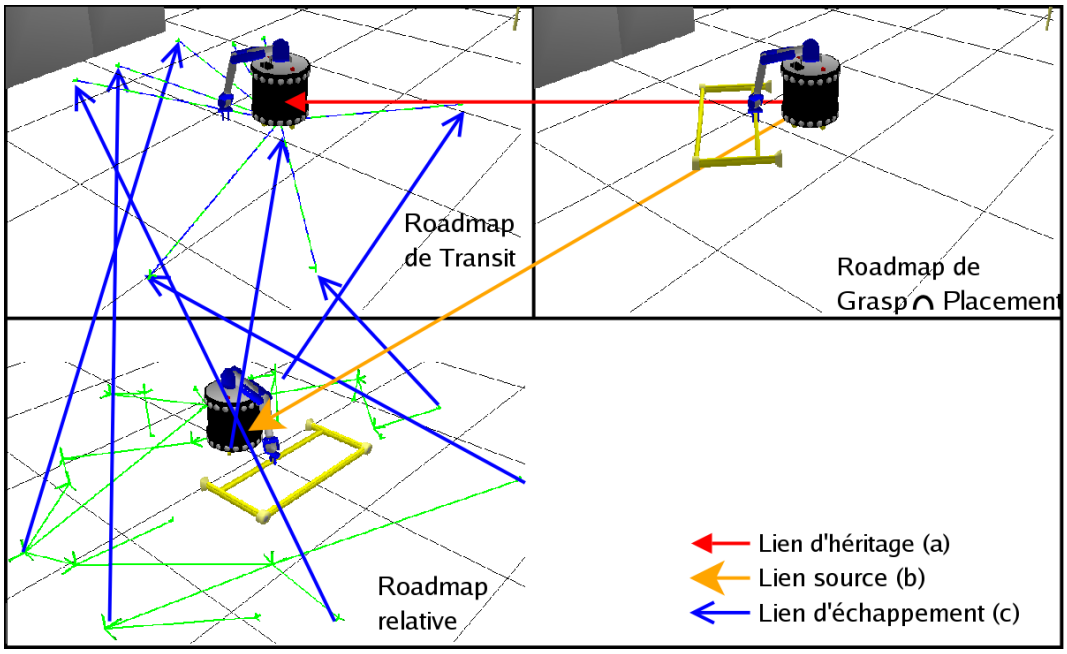


Figure 4.4: Étapes de la création des roadmaps relatives.

Ces roadmaps doivent être liées les unes aux autres par 3 liens d'héritage (§ 3.3) un peu particulier :

- Le *« lien d'héritage d'isolation »* entre la roadmap de *Grasp* ∩ *Placement* et de *Transit* du robot seul (figure 4.4.a).

  Le nœud de la roadmap de *Transit* créé à partir de ce lien ne pourra pas être relié à d'autres nœuds de la roadmap par les méthodes d'apprentissage normales. Il ne pourra être lié qu'aux nœuds trouvés par la roadmap relative (*« nœuds d'échappement »*).
- Le *« lien source »* entre *Grasp* ∩ *Placement* et la roadmap relative. Il permet de définir à la fois la position de l'objet et du robot dans la roadmap relative (figure 4.4.b).

  C'est lui qui permet le calcul de la transformation à appliquer pour passer de la roadmap relative aux positions absolues de la roadmap de *Transit*.
- Le lien d'échappement entre la roadmap de *Transit* et la roadmap relative. Il permet de relier le nœud source ou les *« nœuds d'échappement »* à la roadmap de *Transit* (figure §4.4.c).

  Il utilise la transformation calculé par le *« lien source »* pour reporter les nœuds intéressants dans la roadmap de *Transit*.

Les deux derniers liens sont des liens relatifs, ils possèdent des propriétés de changement de repère. Pour cela nous désignons pour le robot et l'objet un joint sur lequel sera appliqué le changement de repère. Nous utilisons souvent des changements de repère dans le plan. Il faut donc au moins un joint plan pour le robot et l'objet (rappelons que toutes *chaînes cinématiques élémentaires* possèdent 6 degrés de liberté qui représentent son placement).

La procédure d'utilisation des roadmaps relatives est la suivante (figure 4.4) :

1. Dans un premier temps, un nœud dans *Grasp* ∩ *Placement* doit être généré. Celui-ci va activer deux liens : un *« lien d'héritage d'isolation »* vers la roadmap de *Transit* et un *« lien source »*

vers la roadmap relative correspondante.

2. Le premier lien va créer un nœud dans le graphe de *Transit* pour le robot seul juste au moment de la saisie. Ce nœud sera en plus isolé, il sera impossible d'aller en position de saisie par une méthode d'extension classique des roadmaps. Le but est de le relier uniquement aux *« nœuds d'échappements »* trouvés par la roadmap relative.
3. Le second lien crée aussi une configuration pour le robot seul, mais cette fois-ci dans la roadmap relative. Il va créer une référence par rapport à la configuration de l'objet au moment de la saisie et la transformation pour passer de position relative à position absolue. Enfin un lien vers le nœud de transit précédemment créé va être généré.
4. A partir de ce moment, nous avons la première position relative du robot par rapport à l'objet que l'on va étendre par une méthode de diffusion ou lier à une composante déjà créée.
5. Comme décrit précédemment (§ 4.3), nous disposons de deux distances entre lesquelles vont être sélectionnés les nœuds d'échappement.
6. Les nœuds d'échappement vont être reportés dans la roadmap de *Transit* en créant un arc avec le nœud source de transit qui ne sera plus isolé.

Cette méthode permet de limiter le nombre de nœuds qui vont être utilisés pour l'apprentissage de la roadmap de *Transit* aux seuls nœuds d'échappement de la roadmap relative.

Nous avons choisi de représenter toute la complexité de la roadmap relative par l'utilisation d'arcs directs dans la roadmap de *Transit*. Au moment de la validation de la roadmap de *Transit* dans un contexte particulier, il faudra translater la roadmap relative pour remplacer les arcs de la roadmap de *Transit* par des mouvements d'approche parfois complexes de la roadmap relative.

L'utilisation des roadmaps relatives va permettre d'avoir des roadmaps de *Transit* qui seront plus facilement valides quand tous les robots et les objets seront pris en compte. En fait, aSyMov ne peut généralement pas résoudre de problèmes sans l'utilisation de cette technique.

4.4 Le moteur d'enrichissement des roadmaps.

Nous avons décrit 4 types de roadmaps : les roadmaps de mouvements (§ 3.2.1), les roadmaps affinables (§ 3.2.2), les roadmaps heuristiques (§ 4.1) et les roadmaps relatives (§ 4.3). Cela fait donc quatre types d'espaces de recherche, chaque robots et objets pouvant utiliser plusieurs types à la fois. Il faut donc un moyen de décider quelle roadmap étendre.

De plus aSyMov est capable d'utiliser une grande partie des méthodes probabilistes de l'état de l'art tout en incluant de nouvelles méthodes.

Ainsi il faut choisir où et comment étendre les roadmaps sachant que le but est de capturer la topologie du problème à travers les *« Graphes des Cinématiques Élémentaires »* (GCEs § 3.4).

Pour pouvoir étudier les différentes méthodes possibles, nous avons besoin d'un système fortement paramétrable. Pour cela nous avons choisi d'utiliser un ensemble de règles probabilistes. Chaque règle possède un ensemble de préconditions se rapportant généralement à la topologie du problème. Ces

préconditions permettent de pondérer les règles. L'algorithme choisit alors aléatoirement, suivant les pondérations précédemment calculées, une règle à appliquer. Cette règle va ainsi décrire comment étendre les roadmaps.

Une règle se présente sous la forme :
$\mathcal{R}_i$: précondition $\mathrm{P}_{i\ 1}$ et précondition $\mathrm{P}_{i\ 2}$ et ... et précondition $\mathrm{P}_{i\ n}$: Effet $\mathcal{E}_i$

4.4.1 Précondition.

Chaque règle possède un ensemble de préconditions. Pour pouvoir comparer deux règles $\mathcal{R}_1$ et $\mathcal{R}_2$ il faut qu'elles aient des préconditions compatibles.

$\mathcal{R}_1$: précondition $\mathrm{P}_{1\ 1}$ et précondition $\mathrm{P}_{1\ 2}$ et ... et précondition $\mathrm{P}_{1\ n}$: Effet $\mathcal{E}_1$
$\mathcal{R}_2$: précondition $\mathrm{P}_{2\ 1}$ et précondition $\mathrm{P}_{2\ 2}$ et ... et précondition $\mathrm{P}_{2\ n}$: Effet $\mathcal{E}_2$

Il faut que les préconditions $\mathrm{P}_{1\ j}$ et $\mathrm{P}_{2\ j}$ soient comparables. Il existe deux familles de préconditions, celles qui représentent des coûts, c'est-à-dire des probabilités de non sélection et celles qui représentent des gains. Il n'y a pas de grandes différences entre elles hormis le système de calcul des probabilités relatives de chaque règle.

Pour chacune de ces deux familles il existe des types de préconditions différentes qui comparent des éléments différents (temps de calcul de processus, taux d'échec, nombre de nœuds, nombre de composante connexe d'une roadmap ou d'un GCE, nœud avec certaines propriété de connexités entre roadmap, ...). Il n'est pas possible de comparer des temps de calcul avec des nombres de nœuds. C'est pour cela que nous avons choisi des types différents qui regroupent les données comparables entre elles.

Voici un ensemble de types de préconditions que nous avons développés. Notons que le système permet d'ajouter de nouveaux types assez facilement :

- Coût de taille de roadmap : coût dépendant du nombre de nœuds de la roadmap, où du nombre de nœuds rejetés par la méthode de visibilité (§ 2.3.2).
  Ce coût permet d'estimer la couverture de l'espace de configuration représentée par la roadmap. Nous savons (§ 2.8.4) qu'il est intéressant d'explorer *Grasp* ∩ *Placement* avant *Transit* et *Transfert*. Ce coût permet d'estimer quand *Grasp* ∩ *Placement* est suffisamment développé.
  Une autre utilisation est la restriction de la taille d'une roadmap par rapport à une autre. Par exemple la roadmap des configurations de prise par rapport à la roadmap de *Transfert*.
- Intérêt symbolique : valeur d'intérêts provenant d'un planificateur symbolique. Les actions d'un plan heuristique vont influencer l'intérêt de cette précondition. Par exemple, si une action de *Transfert* est prévue, il est intéressant d'accroître l'intérêt d'une règle qui enrichie la roadmap de *Transfert* concernée.
  Nous verrons cela au chapitre 6.
- Intérêt des nœuds spécifiques : permet de sélectionner un certain nombre de nœuds ayant des caractéristiques communes et de calculer un intérêt en fonction de leur nombre.
  Cela permet de sélectionner les nœuds sources d'une roadmap relative qui ne sont pas encore

liés à des nœuds d'échappement. Il peut être intéressant d'étendre la roadmap relative pour leur trouver des nœuds d'échappement.

- Intérêt de connexité : permet de sélectionner des paires de nœuds (N_1, N_2) suivant des critères de connexité.

  Son principal intérêt est de pouvoir comparer la connexité au sein d'une roadmap par rapport à la connexité au niveau des GCEs. On peut notamment représenter l'exemple figure 4.1 et comparer la connexité de la roadmap heuristique de mouvement de l'objet seul avec le GCE du problème *« un robot et un objet »*.

  Dans le cas où les nœuds N_1 et N_2 ne sont pas liés à travers les GCEs il peut être intéressant de tenter de les relier entre eux. S'ils appartiennent à la même composante connexe de la roadmap heuristique alors il est intéressant de rechercher une position intermédiaire et d'enrichir la roadmap de *Transit*. Sinon il est intéressant de trouver une configuration de prise commune et d'enrichir la roadmap de *Transfert*.

En plus du type auquel elles appartiennent les préconditions ont systématiquement une fonction de calcul de leur coût ou de leur intérêt et une fonction de sélection de nœuds.

Certains types de précondition font une sélection d'un certain nombre de nœuds pour calculer leur coût. Or ces nœuds sont souvent intéressant et utilisables pour guider les méthodes d'exploration des roadmaps. Il est donc intéressant de récupérer ces nœuds.

Par exemple, une précondition *« Intérêt des nœuds spécifiques »* permet de sélectionner des nœuds sources d'une roadmap relative qui doivent être reliés à des nœuds d'échappement. Ces nœuds peuvent servir de nœuds initiaux pour des méthodes RRT (§ 2.4.1).

Les paramètres des préconditions sont propres à chaque type. Il y a souvent besoin de spécifier la roadmap étudiée, mais il est aussi possible de prendre les ensembles de nœuds retournés par d'autres préconditions en paramètre d'entrée (figure 4.5.a).

4.4.2 Les effets.

Les effets des règles sont de deux types : la sélection d'un sous-ensemble de règles ou la sélection d'une roadmap et d'une méthode d'expansion.

Le développement en un sous-ensemble de règles permet d'avoir une représentation hiérarchique des règles et de limiter le nombre de préconditions à calculer. Si une règle R est choisie et possède des sous-règles, il faut alors calculer les préconditions des sous-règles et choisir l'une d'entre elles.

La sélection de la méthode d'expansion permet de choisir une grande variété de paramètres, notamment :

1. La roadmap à étendre.
2. Les liens d'héritages qui vont être automatiquement activés (et donc générer des nœuds dans des sous-roadmaps).

   Ainsi l'ajout d'un nœud dans $Grasp \cap Placement$ pourra automatiquement créer un nœud de *Transfert*.

3. Les liens d'héritages qui vont être utilisés pour la recomposition des nœuds (§ 4.2).

   Par exemple l'utilisation d'une configuration de prise d'un nœud d'une roadmap heuristique pour créer un nœud de *Grasp* ∩ *Placement*.

4. La méthode d'enrichissement utilisée (RRT, visibilité, recherche près d'un nœud ...).

5. Le critère d'arrêt : le nombre de nœud, la connexité entre plusieurs nœuds, le nombre d'échecs, ... (figure 4.5.f).

   Généralement le critère d'arrêt est faible puisque l'on va tenter plusieurs règles. Ainsi il est possible de limiter l'extension à 2 ou 3 nœuds avant d'évaluer à nouveau les règles.

6. La précondition à utiliser pour instancier les nœuds à utiliser.

   Il est par exemple possible de chercher à connecter 2 nœuds N_1, N_2 de *Transit* qui sont liés à deux nœuds de *Grasp* ∩ *Placement* N_3, N_4 qui eux même ne sont pas dans la même composante connexe mais sont liés à une unique position de l'objet N_5.

   Il est possible d'utiliser une précondition du type *« Intérêt de connexité »* pour comparer la connexité de *Grasp* ∩ *Placement* avec la roadmap de *Placement* (constitué de nœuds isolés) pour sélectionner le couple (N_3, N_4) et par extension le couple (N_1, N_2) dans *Transit*. De plus le nombre de paires trouvées représentera bien l'intérêt d'une telle règle.

4.4.3 Le choix.

Le choix d'une règle dans un ensemble de règles donné se fait en 4 étapes :

1. Calcul des valeurs des différentes préconditions.

2. Calcul des probabilités associées. Elles sont proportionnelles (gain) ou inversement proportionnelles (coût) aux valeurs des préconditions. Elles sont normalisées sur l'ensemble des règles.

3. Calcul des probabilités des règles.

   Hypothèse : *Nous supposons pour simplifier le calcul que les valeurs des préconditions d'une même règle sont décorrélées.*

   Ainsi la probabilité $P(\mathcal{R}_i)$ de sélection de la règle est le produit des probabilités de chaque règle. Et la probabilité de sélectionner une règle applicable est :

   $$P(\mathcal{R}_i applicable / applicable) = \frac{P(\mathcal{R}_i applicable)}{\sum_j P(\mathcal{R}_j applicable)}$$

   avec $P(\mathcal{R}_i applicable) = applicable(\mathcal{R}_i).\prod_j P(P_{j\,i})$

   Où *applicable* est une fonction qui retourne 1 si les préconditions fournissent assez de paramètres pour appliquer l'effet et 0 sinon. Un *« bi-RRT »* a besoin d'au moins une paire de nœuds qui sera fourni par une précondition. Si celle-ci ne trouve aucune paire alors la règle n'est pas applicable.

4. Tirage aléatoire d'une règle suivant les probabilités obtenues à l'étape 3. Si c'est un effet de développement en sous-règles on recommence au numéro 1 avec le sous-ensemble de règles, sinon on étend la roadmap précisée par l'effet de la règle.

L'algorithme peut recommencer ce choix un certain nombre de fois ou s'arrêter en fonction d'un critère de connexité au niveau des GCEs.

4.4.4 L'exemple de la manipulation.

Nous allons maintenant voir les règles utilisées sur l'exemple du bras manipulateur avec la barre prise dans une cage (figure 4.1). Cet exemple a été présenté dans la thèse d'Anis Sahabani [Sahbani 03].

Nous avons déjà vu qu'il y avait une compatibilité entre les GCEs et le graphe de manipulation pour un robot et un objet. Cette compatibilité s'étend aussi au systèmes de règles.

L'algorithme de la manipulation prévoit des choix stratégiques probabiliste pour choisir quelle roadmap étendre § 2.8.4.

Le premier choix se fait entre étendre $Grasp \cap Placement$ ou chercher des chemins de *Transit* et de *Transfert*. Ce choix se fait proportionnellement au nombre d'échecs de la méthode de visibilité. Cela correspond à une précondition du type *« coût de taille de roadmap »*.

Ce premier choix peut être donc vu comme un système de deux règles :

	« Coût de taille de roadmap »	Effets:
$\mathcal{R}_1$	échecs visibilité dans $Grasp \cap Placement$	Enrichir $Grasp \cap Placement$
$\mathcal{R}_2$	coût fixe	Extension *Transit*/*Transfert*

La méthode d'enrichissement de $Grasp \cap Placement$ est celle développé par Sahabani [Sahbani 03]. C'est une méthode par visibilité sauf qu'elle teste la validité d'une méthode locale en commençant par tester l'objet seul. Si le mouvement est faisable pour l'objet seul et non faisable dans $Grasp \cap Placement$ alors elle cherche une position de pose intermédiaire.

L'extension *Transit*/*Transfert* sélectionne un nœud de $Grasp \cap Placement$ et essai de le lier à des nœuds de $Grasp \cap Placement$ appartenant à d'autres composantes du graphe de manipulation. Ce lien se faisant par un couple de trajectoires (*Transit*, *Transfert*) ou (*Transfert*, *Transit*).

Nous avons modifié ces règles pour remplacer l'extension *Transit*/*Transfert* par le sous-ensemble de règles suivant :

	« Intérêt de connexité »	Effets:
$\mathcal{R}_1$	$N_1, N2 \in Grasp \cap Placement$ N_1 et N_2 non connexe dans les GCEs N_1 et N_2 connexe dans la roadmap heuristique de l'objet	Enrichir *Transfert*
$\mathcal{R}_2$	$N_1, N2 \in Grasp \cap Placement$ N_1 et N_2 non connexe dans les GCEs N_1 et N_2 connexe dans *Placement*	Enrichir *Transit*

Cela permet d'utiliser la topologie de la roadmap heuristique. Les positions intermédiaires dans *Placement* étant trouvées par la méthode d'enrichissement de $Grasp \cap Placement$.

De même « Enrichir *Transfert* » peut être détaillé pour avoir des règles qui vont créer ou réutiliser des configurations de prises d'une roadmap heuristique.

« Enrichir *Transit* » peut lui aussi faire des recherche dans *Transit* ou dans la roadmap relative à l'objet déplaçable. La figure 4.5 montre l'interface d'implémentation du système de règles et notamment les sous-règles de « Enrichir *Transit* ».

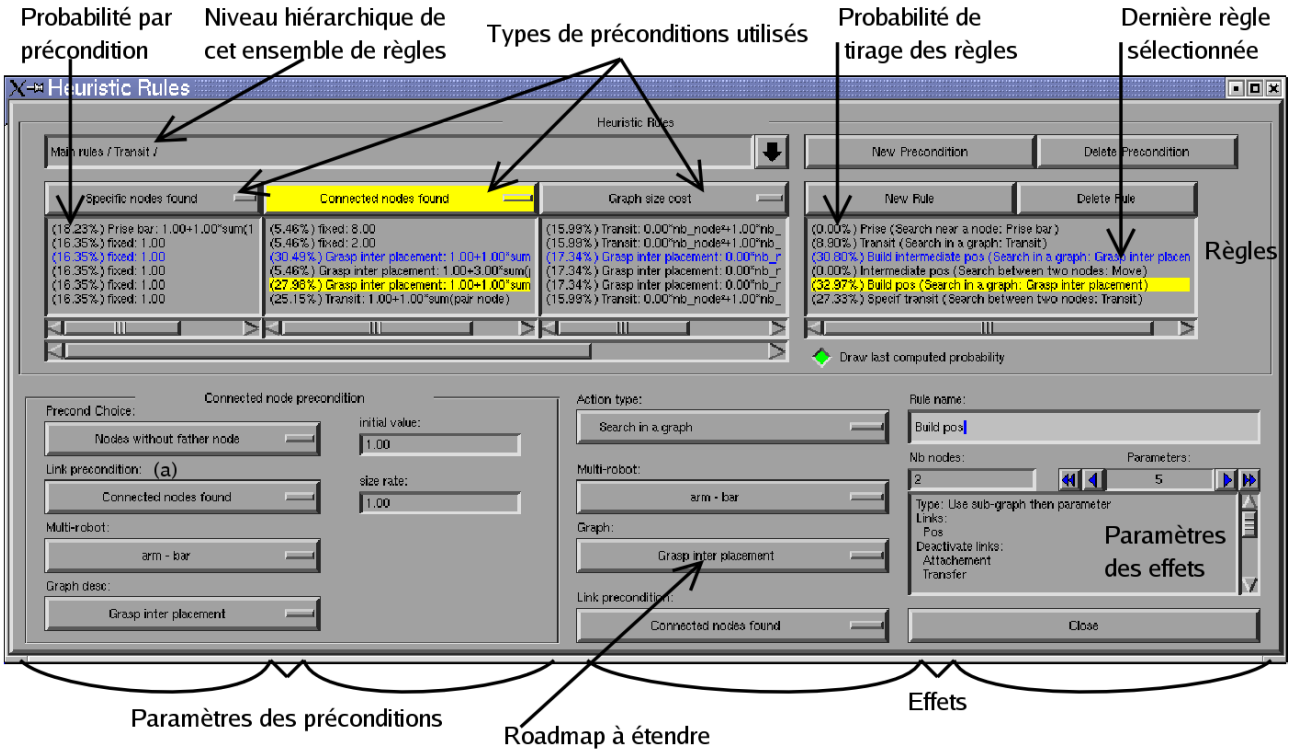


Figure 4.5: Système de règles implémenté pour le problème de manipulation un robot et un objet. Sous ensemble correspondant à une liaison par un chemin de transit.

Les règles présentées ci-dessus permettent d'étendre les roadmaps pour résoudre le problème de la manipulation avec un robot et un objet déplaçable pour des positions de pose et des prise continues. Il est possible de multiplier le nombre de ces règles pour prendre en compte plusieurs robots et plusieurs objets.

Remarque :
Tout au long de cette thèse, nous nous sommes intéressés au développement des méthodes d'enrichissement des roadmaps et aux paramètres qui jouent sur la combinaison de ces méthodes. C'est pourquoi nous avons développé un système fortement paramétrable. Toutefois, nous pensons qu'il est possible pour une classe donnée de problèmes de générer automatiquement ces paramètres. Cela pourra être effectué dans de futurs travaux.

4.5 Conclusion.

Nous avons développé une méthodologie permettant d'utiliser les résultats de l'état de l'art et de les combiner pour aborder des problèmes plus complexe.

Toutefois, hormis les roadmaps relatives, aucune roadmap ne prend en compte les autres robots ou objets. Autrement dit, un objet qui rompt la connexité d'une composante connexe n'a pas d'influence ni sur les GCEs ni sur les méthodes d'apprentissage des roadmaps.

Comme nous l'avons déjà fait remarquer, pour parer à ce problème il est possible de prendre en compte explicitement les configurations des autres robots ou objets. Mais dans ce cas nous augmentons très fortement la complexité des roadmaps.

La solution que nous avons choisie est de prendre en compte un contexte. Ce contexte désignera les configurations des autres robots et objets. Les arcs de la roadmap auront une liste de configurations d'autres robots pour lesquelles ils ont été validés ou invalidés. Ceci permet de connaître l'état de la roadmap dans différents contextes.

Il ne reste plus qu'à choisir de bons contextes qui vont servir lors de la recherche d'un plan valide. Un choix aléatoire reviendrait à l'utilisation d'une roadmap portant sur la concaténation des configurations des robots. Pour éviter cette complexité, il faut se limiter aux contextes qui ont de grandes chances d'apparaître dans le plan solution.

Dans le chapitre précédent, nous avons remarqué qu'une connexité au niveau des GCEs ne suffisait pas à prouver l'existence d'une solution. Il faut en plus une étape de validation qui doit construire un plan solution.

C'est de ce plan dont on doit se servir pour établir les bons contextes à utiliser pour faire l'apprentissage des roadmaps.

Nous avons décidé que pour la recherche de ce plan nous utiliserons des techniques de recherche dans l'espace d'état issue de la planification de tâches. Ainsi cette étape de planification devra non seulement servir à trouver un chemin dans les GCEs mais aussi à les construire en prenant en compte les autres robots et objets tels qu'ils apparaissent dans l'élaboration du plan.

Le chapitre suivant va justement présenter les techniques de planifications de tâches et comment les faire cohabiter avec la représentation géométrique du monde que nous avons développée dans le chapitre précédent.

Chapitre 5

ASyMov et la planification de tâches.

Dans cette thèse nous avons décidé d'élaborer un planificateur « *aSyMov* » [1] nous permettant de réaliser des tâches de manipulation prenant en compte explicitement la géométrie de l'environnement.

Les deux chapitres précédents nous ont permis de définir une représentation de l'espace de recherche et des méthodes pour explorer la topologie du problème. Mais ils ont tous les deux indiqué la nécessité de représentation complémentaire de l'état du monde. Une recherche de solutions dans l'espace d'état permettrait à la fois de valider la topologie trouvée et de faire un apprentissage des roadmaps qui prenne en compte tous les robots et les objets de l'environnement.

Il se trouve justement que les techniques de planification de tâches permettent d'explorer un espace d'état abstrait. Il peut être intéressant de s'inspirer de ces techniques. Nous avons alors décidé de franchir une étape supplémentaire en décidant d'implémenter complètement les techniques de planification de tâches pour permettre la résolution de problèmes prenant en compte non seulement la géométrie de l'environnement mais aussi des informations symboliques. Il sera par exemple possible d'indiquer qu'il faut une clef pour ouvrir une porte, qu'il faut mettre de la colle avant d'assembler des objets, qu'un robot doit prendre des photos sur des zones données, etc.

Pour résoudre un problème de planification, il est souvent impensable de parcourir entièrement l'espace d'état pour trouver un plan qui satisfasse le but recherché. De nouvelles techniques ont récemment relancé la course au meilleur planificateur, à tel point qu'un langage de description commun : « *PDDL* » [M. Ghallab 98], et qu'une compétition, liée depuis 1998 aux conférences AIPS, ont vu le jour.

Dans un premier temps nous allons présenter le problème de la planification de tâches avec le langage de description de domaine « *PDDL* ». Puis nous présenterons brièvement les différentes tech-

[1] a Symbolic Move3D

niques utilisées pour résoudre ces problèmes ainsi que quelques limitations. Enfin nous montrerons un moyen original de combiner les techniques géométriques développées dans les chapitres précédents et les techniques symboliques de la planification de tâches.

5.1 Le domaine de la planification de tâches.

L'objectif de la planification de tâches et de trouver un ordonnancement d'actions (plan) qui est exécuté depuis un état initial de l'environnement et permet d'atteindre un ensemble de buts.

Le problème se définit donc par le triplet $< \mathcal{E}_{init}, \mathcal{A}, \mathcal{B}_{ut} >$ où $\mathcal{E}_{init}$ représente l'état dans lequel se trouve l'environnement au moment de l'exécution du plan, $\mathcal{A}$ modélise l'ensemble des actions α que l'agent peut exécuter afin de réaliser les buts définis par $\mathcal{B}_{ut}$.

Avant de s'intéresser au problème algorithmique de la recherche de plan solution, il faut définir un modèle pour représenter le monde ainsi que les changements que les agents peuvent décider d'appliquer. Nous allons maintenant présenter le formalisme *« STRIPS »* et son extension *« PDDL »* dont nous allons nous servir tout au long de cette thèse.

5.1.1 Le formalisme « STRIPS ».

Le but de la planification de tâches est d'exécuter un enchaînement d'actions, il faut être capable de décrire l'état du monde $\mathcal{E}_i$ qui résulterait de l'application d'une action α sur un état $\mathcal{E}_{i-1}$. Pour cela il est nécessaire de préciser ce qui a changé et ce qui reste inchangé. Généralement ce qui change avec l'application d'une action n'est qu'une petite partie de l'état. Il paraît donc intéressant de ne représenter pour les actions que les changements apportés à l'état.

Le formalisme *« STRIPS »* [2] [Fikes 71] permet de représenter de manière concise cette évolution d'état après application d'une action. Les états sont modélisés par une conjonction de faits (en logique propositionnelle) réduite grâce à l'utilisation de l'hypothèse du monde clos : seul les faits positifs sont conservés dans la description d'un état.

Pour représenter un changement, *« STRIPS »* va donc décomposer les modifications en deux : ce qui est devenu vrai et ce qui est devenu faux après application de l'action. Le modèle d'une action α se décompose alors en trois parties :

- P_α : conjonction de faits qui représente les préconditions de l'action α, i.e. ce qui doit être vrai pour appliquer l'action α dans l'état $\mathcal{E}$: $P_\alpha \subseteq \mathcal{E}$.
- D_α : conjonction de faits qui représente ce qui n'est plus vrai après l'application de l'action α (*« delete list »*) : $D_\alpha \subseteq P_\alpha$.
- A_α : conjonction de faits qui représente ce qui deviendrait vrai par l'application de l'action α (*« add list »*) : $(A_\alpha \cap P_\alpha) = \emptyset$.

Cette représentation simpliste du monde peut parfois rendre délicate la modélisation d'un domaine particulier. Toutefois ce formalisme reste le plus utilisé par les planificateurs actuels.

[2] STanford Research Institute Problem Solver

5.1.2 Le formalisme « PDDL ».

Le formalisme *« STRIPS »* étant plutôt pauvre, la plupart des planificateurs ont proposé des extensions. Dans le but de comparer les performances des algorithmes le langage de description de problèmes *« PDDL »*[3] [M. Ghallab 98] et sa version 2.1 [Fox 01] unifient les différentes représentations ne conservant que les extensions majeures.

Il permet notamment :

- le typage des atomes des faits qui réduit le nombre d'instanciations possible pour les actions.
- les actions conditionnelles (opérateur « `or` » dans les préconditions) qui permet de regrouper sous une même action différents cas de figures.
- le traitement des variables numériques qui permet la représentation de ressources telles qu'une batterie, un conteneur, ...
- la représentation explicite du temps et des durées.
- une sémantique pour représenter les contraintes temporelles.
- une spécification d'une métrique pour le plan.

La plupart des planificateurs ne prennent pas en compte la totalité des représentations possibles de *« PDDL »*. Pour notre part, nous ne nous intéresserons pas aux trois dernières fonctions présentées ci-dessus.

Remarque :
Il est vrai qu'il serait intéressant de lier la représentation géométrique avec une représentation du temps pour avoir des actions de déplacement dont la durée dépend des chemins trouvés dans l'environnement. Mais cette fonction est assez complexe à mettre en place. La plupart des planificateurs utilisent des durées qui sont au mieux des fonctions linéaires par morceau de variables d'état. L'élaboration d'un planificateur de tâches permettant de traiter des problèmes avec des durées qui sont des fonctions quelconques n'entre pas dans le cadre de cette thèse.

De plus la planification de mouvement probabiliste suppose la construction d'une roadmap puis son optimisation. La longueur des trajectoires données par une roadmap est toujours supérieure à sa longueur optimisée. L'utilisation des roadmaps au cours de la planification peut donc sur-contraindre le problème.

5.1.3 Le langage « PDDL ».

La définition d'un problème dans le format *« PDDL »* se décompose en deux parties : le *« domaine »* et le *« problème »*. Le *« domaine »* définit entre autre les types, les prédicats et les actions possibles, le *« problème »* définit pour sa part l'état initial et le but du problème.

[3] *Planning Domain Definition Language*

5.1.3.1 La définition du « domaine ».

Le format de la définition d'un *« domaine »* (sans prise en compte du temps) est[4] :

```
(define (domain DOMAIN_NAME)
  (:requirements [:strips] [:equality] [:typing] [:adl] [:fluents])
  (:types TYPE_1_NAME ...)
  (:constants A1 [- TYPE] ... AN [- TYPE])
  (:predicates (PREDICATE_1_NAME [?A1 [- TYPE] ... ?AN [- TYPE]])
               (PREDICATE_2_NAME [?A1 [- TYPE] ... ?AN [- TYPE]])
   ...)
  (:functions (FUNCTION_1_NAME [?A1 [- TYPE] ... ?AN [- TYPE]])
              (FUNCTION_2_NAME [?A1 [- TYPE] ... ?AN [- TYPE]])
   ...)

  (:action ACTION_1_NAME
   [:parameters (?A1 [- TYPE] ... ?AN [- TYPE])]
   [:precondition PRECOND_FORMULA]
   [:effect EFFECT_FORMULA]
  )
 (:action ACTION_2_NAME
  ...)
...)
```

:requirements

Puisque *« PDDL »* est un langage très général la plupart des planificateurs ne peuvent prendre en compte qu'un sous-ensemble de ce langage. « `requirements` » permet de spécifier dans le *« domaine »* quels sont les parties du langage *« PDDL »* qui doivent être supportés. Les sous-ensembles de *« PDDL »* les plus généralement utilisés sont :

`:strips` qui représente le formalisme *« STRIPS »* (§ 5.1.1).

`:equality` qui représente l'utilisation de l'opérateur « `=` ».

`:typing` : le domaine emploie des types.

`:adl` : le domaine emploie l'opérateur de branchement conditionnel « `or` ».

`:fluent` : le domaine utilise des variables numériques.

`:durative-actions :duration-inequalities` : le domaine a une représentation du temps.

:types

Cela définit les noms des types qui pourront être utilisés pour restreindre l'instanciation des actions et des faits.

[4] « `[]` » représente les paramètres facultatifs, « `?Ai - TYPE` » désigne un atome quelconque de type « `TYPE` » défini par « `:types` ».

:constants
Cela définit des atomes (typés) qui pourront être utilisés dans tous les problèmes basés sur ce domaine. Cela permet aussi d'utiliser ces atomes directement dans la définition des actions.

:predicates
Cela définit le format des faits que peut avoir un état. Un fait est *« vrai »* dans l'état $\mathcal{E}$ s'il existe dans $\mathcal{E}$, sinon il est *« faux »*.

:functions
Similaire à « `:predicates` », à ces faits sont associée une valeur numérique. Ainsi `(:functions (niveau-essence ?v - vehicule) )` permet de créer un fait `(niveau-essence voiture1)` auquel sera associé une valeur numérique qui changera en fonction des actions. Cela permet la création de ressources.

:action
Permet de définir les actions. Les préconditions « `PRECOND_FORMULA` » sont définies au choix par :

- `(PREDICATE ARG_1... ARG_N)`
- `(and PRECOND_FORMULA_1 ... PRECOND_FORMULA_N)` : ces deux définitions sont utilisées pour le sous-ensemble « `:strips` » de *« PDDL »*.
- `(not PRECOND_FORMULA)`
- `(or PRECOND_FORMULA_1 ... PRECOND_FORMULA_N)` : ces deux définitions sont utilisées avec « `:adl` ». « `:adl` » permet aussi d'autres possibilités (« `forall` », « `exists` ») mais nous ne les utiliserons pas dans nos exemples.
- `(= ARG_1 ARG_2)]` : utilisé avec « `:equality` ».
- `(=|<=|<|>=|> FORMULE_1 FORMULE_2)`[5] : utilisé avec « `:fluent` ».
  Où « `FORMULE` » est égale à « `(FUNCTION ARG_1 ... ARG_N)` »
  ou à « `(-|+|*|/ FORMULE_1 FORMULE_2)` ».

Les effets « `EFFECT_FORMULA` » sont définis par :

- `(PREDICATE ARG_1 ... ARG_N)` : un fait ajouté (A_α pour *« STRIPS »*).
- `(not (PREDICATE ARG_1 ... ARG_N))` : un fait supprimé (D_α pour *« STRIPS »*).
- `(and EFFECT_FORMULA_1 ... EFFECT_FORMULA_N)`

- `(increase (FUNCTION ARG_1 ... ARG_N) FORMULE)`
- `(decrease (FUNCTION ARG_1 ... ARG_N) FORMULE)`
- `(assign (FUNCTION ARG_1 ... ARG_N) FORMULE)` : trois opérations pour modifier les variables d'état numérique avec (`:fluent`).

[5] « `a|b` » signifie qu'il faut choisir « `a` » ou « `b` »

5.1.3.2 La définition du « problème ».

Il est défini par :

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects ATOM_1 [- TYPE] ... ATOM_N [- TYPE])
  (:init FACT_1 ... FACT_N)
  (:goal CONDITION_FORMULA)
)
```

:domain
Il définit le domaine attaché au problème (§ 5.1.3.1).

:objects
Définit les atomes du problème qui peuvent être typés comme « `voiture1 - vehicule` ».

:init
Définit l'état initial du problème. « `FACT` » peut valoir « `(PREDICATE_NAME OBJ_i... OBJ_k)` » pour définir un fait ou « `(= (FUNCTION_NAME OBJ_i... OBJ_k) nombre)` » pour initialiser une variable d'état numérique.

:goal
Définit l'ensemble des faits buts du problème. Il a la même sémantique que les préconditions d'une action.

5.1.4 L'exemple du domaine des cubes.

Le domaine des cubes est un des premiers exemples utilisés en planification de tâches. Le but est de construire un ensemble de tours en déplaçant les cubes à partir d'un état connu.

Bien qu'éprouvé depuis longtemps, ce problème reste tout de même intéressant par la taille de son espace de recherche. Pour n cubes il existe $n!$ façons de les agencer. Nous allons nous servir de ce domaine pour illustrer les techniques de planification que nous allons développer ci-dessous. La représentation de son *« domaine »* en *« PDDL »* est la suivante :

```
(define (domain Cubes)
  (:requirements :strips)
  (:types cube)
  (:predicates (Sur ?X - cube ?Y - cube) ;; le cube ?X est posé sur le cube ?Y
               (Sur_Table ?X - cube)     ;; le cube ?X est posé sur la table
               (Libre ?X - cube)         ;; il n'y a pas de cube sur ?X
  )
```

```
  (:action Poser        ;; déplace le cube ?X posé sur ?Y vers la table
   :parameters (?X - cube ?Y - cube)
   :precondition (and (Libre ?X) (Sur ?X ?Y))
   :effect (and (Sur_Table ?X) (Libre ?Y)
                (not (Sur ?X ?Y)))
  )
  (:action Met_Sur       ;; déplace le cube ?X posé sur la table vers ?Y
   :parameters (?X - cube ?Y - cube)
   :precondition (and (Libre ?X) (Sur_Table ?X) (Libre ?Y))
   :effect (and (Sur ?X ?Y)
                (not (Sur_Table ?X)) (not (Libre ?Y)))
  )
  (:action Deplace       ;; déplace le cube ?X posé sur ?Y vers ?Z
   :parameters (?X - cube ?Y - cube ?Z - cube)
   :precondition (and (Libre ?X) (Sur ?X ?Y) (Libre ?Z))
   :effect (and (Libre ?Y) (Sur ?X ?Z)
                (not (Sur ?X ?Y)) (not (Libre ?Z)))
  )
)
```

5.2 Les techniques de la planification de tâches.

Après avoir défini la manière de représenter l'état du monde et les actions applicables, nous allons maintenant montrer les techniques qui permettent de rechercher un plan solution. Des planificateurs de tâches ont tous une représentation explicite ou implicite de l'état du monde. Ils vont donc chercher à explorer l'espace de ces états. Le but des techniques présentées ici est de réduire l'exploration de l'espace d'état.

5.2.1 Planifier dans l'espace des plans partiels.

La planification dans l'espace des plans partiels n'a pas de représentation explicite des états, mais elle va construire un graphe dont les nœuds sont des actions et les arcs des relations de précédences.

Nous trouvons dans cette catégorie des planificateurs tels que : « *TWEAK* » [Chapman 87], « *SNLP* » [McAllester 91] et « *IxTeT* » [Ghallab 89, Laborie 95] qui est capable de gérer aussi les ressources et le temps.

Ce graphe représente un plan temporaire qui possède un certain nombre de défauts que l'algorithme va tenter de corriger. Cette correction se fait de manière incrémentale et est basée sur le principe de moindre engagement. Pour résoudre un défaut, le planificateur va ajouter une action de telle manière que ses effets suppriment le défaut et en rajoutent le moins possible.

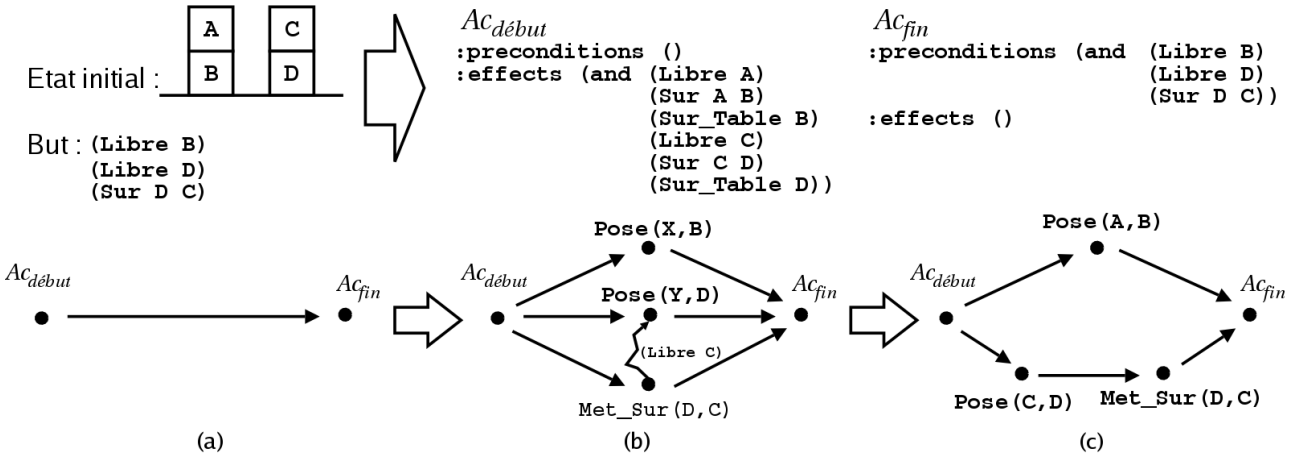


Figure 5.1: Résolution du problème des cubes dans l'espace des plans partiels. (b) fait apparaître une *« menace »* résolue par *« promotion »* de `Met_Sur(D,C)`

.

La figure 5.1 montre la résolution d'un problème dans le monde des cubes. Deux actions $Ac_{début}$ et Ac_{fin} permettent de représenter l'état initial et le but du problème. Le plan (a) n'est pas valide, aucune précondition de Ac_{fin} n'est satisfaite. Il va donc falloir appliquer un certain nombre d'actions avant Ac_{fin} dont les effets vont permettre de satisfaire les préconditions de Ac_{fin}.

Ces actions vont être partiellement instanciées (figure 5.1.b) afin d'autoriser plus de flexibilité. A ce stade de la recherche, les trois actions peuvent s'exécuter en parallèle. Mais si `Y` vaut `C` ce parallélisme n'est plus possible. On dit alors qu'il existe une *« menace »* entre les actions `Poser(Y,D)` et `Met_Sur(D,C)` qu'il faut supprimer. Une menace peut être résolue de trois façons différentes :

- la *« promotion »* de `Poser(Y,D)` : l'action se synchronise après `Met_Sur(D,C)`.
- la *« démotion »* de `Poser(Y,D)` : l'action se synchronise avant `Met_Sur(D,C)`.
- la séparation : une contrainte d'inégalité est ajoutée (`Y` $\neq$ `C`).

Cette technique de planification a pour gros avantage de fournir un plan parallèle. La plupart des planificateurs dans l'espace d'état ne fournissent qu'un plan séquentiel.

Il existe des techniques pour paralléliser un plan séquentiel [Fade 90], mais celles-ci reviennent en fait à repasser dans l'espace des plans partiels à partir d'un plan séquentiel trouvé. L'espace des plans partiels est donc souvent utilisé pour exprimer le plan final pour faciliter sa correction en cas d'événements inattendus, pour l'optimiser ou pour calculer des événements de synchronisation quand il y a une exécution parallèle.

D'un point de vue géométrique, nous pouvons faire une analogie entre les GCEs et le graphe des plans partiels. Toutefois dans les GCEs les menaces entre actions dues au risque de collision ne sont pas prises en compte. Malheureusement pour pouvoir vérifier la validité d'une action de déplacement il faut en toute rigueur connaître l'état géométrique complet du système. Les algorithmes géométriques que nous avons développés dans les chapitres précédents utilisent déjà les hypothèses de la planification

dans l'espace des plans partiels, ils ont maintenant besoin d'être confronté à l'espace d'état pour être validé.

5.2.2 Réduire les possibilités.

L'exploration de l'espace d'état est une opération coûteuse, que l'on commence par le but ou par l'état initial, il y aura des états explorés inutilement. Si l'on part de l'état initial, nous pouvons explorer des états qui ne mèneront pas au but et vice versa (figure 5.2.b et c).

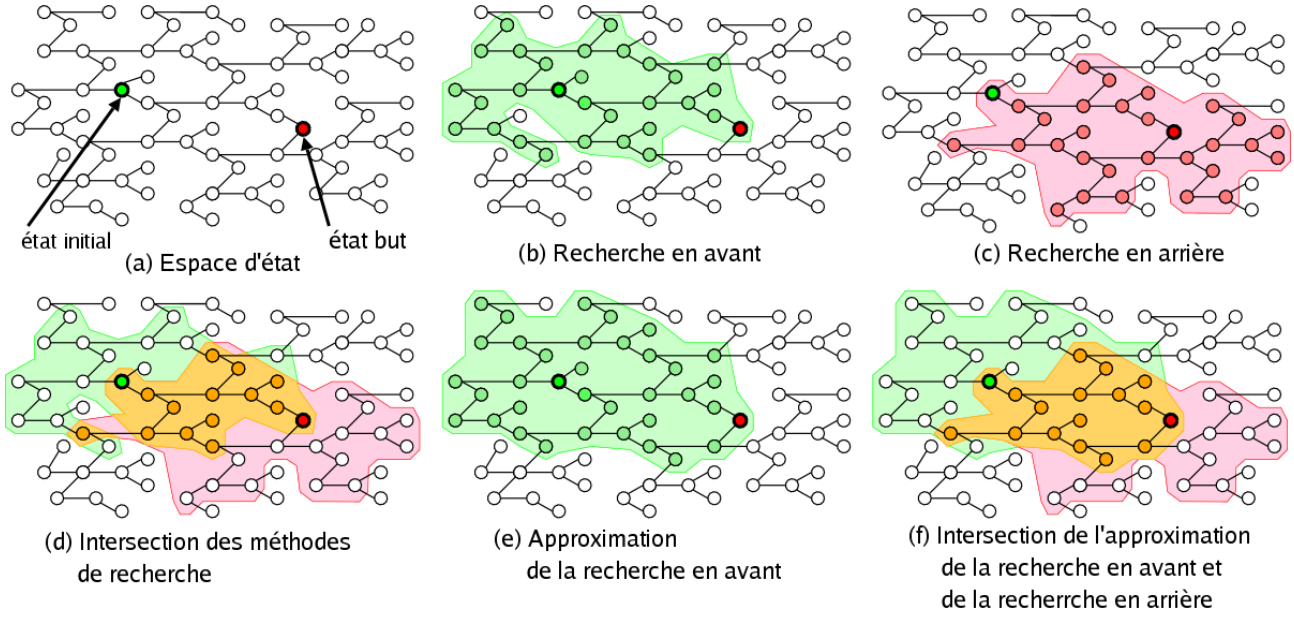


Figure 5.2: Restriction de l'espace de recherche par combinaison des méthodes de recherche à partir de l'état initial et à partir de l'état but.

Une idée pour réduire le nombre d'états explorés est d'étudier les états résultant de l'intersection de ces deux recherches (figure 5.2.d). Toutefois pour utiliser simultanément les deux informations, il faut les connaître en tout point du graphe de recherche, y compris au niveau du but. Il faudrait alors avoir déjà résolu le problème.

Une autre idée possible est de faire dans un premier temps une recherche approximative et rapide du problème puis d'utiliser cette information pour restreindre la recherche dans l'autre sens (figure 5.2.e et f).

C'est sur cette idée qu'est basé le planificateur « *GraphPlan* » [Blum 97] et ses descendants « *IPP* » [Koehler 98] et « *STAN* » [Fox 99]. Il effectue une recherche à partir de l'état initial pour construire une approximation du graphe d'accessibilité qui peut être construit en temps polynomial. Il utilise alors une recherche en arrière dans l'espace des plans partiels mais en ne choisissant que les actions qui ont un état de départ inclus dans le graphe d'accessibilité.

La construction du graphe d'accessibilité est basée sur le regroupement des faits qui peuvent être vrais à un niveau donné. Ce niveau correspond au nombre d'actions qui ont été appliquées. La figure 5.3 représente le regroupement des faits pour un problème de trois cubes. Nous pouvons remarquer

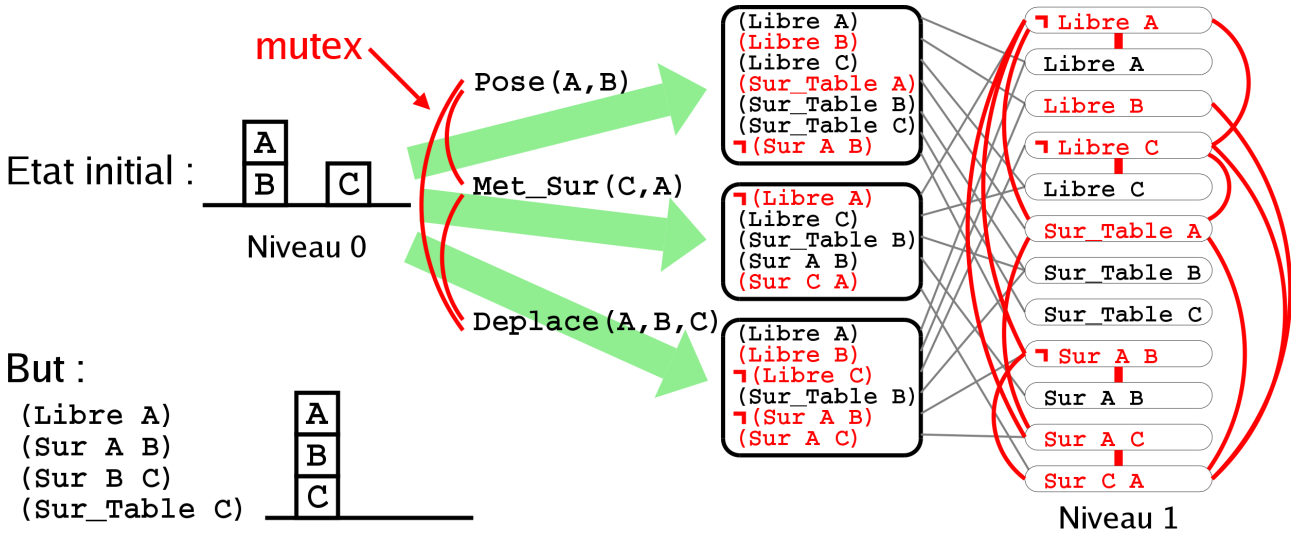


Figure 5.3: Regroupement des faits pour la construction du graphe d'accessibilité sur le problème des cubes avec création des « *mutex* ».

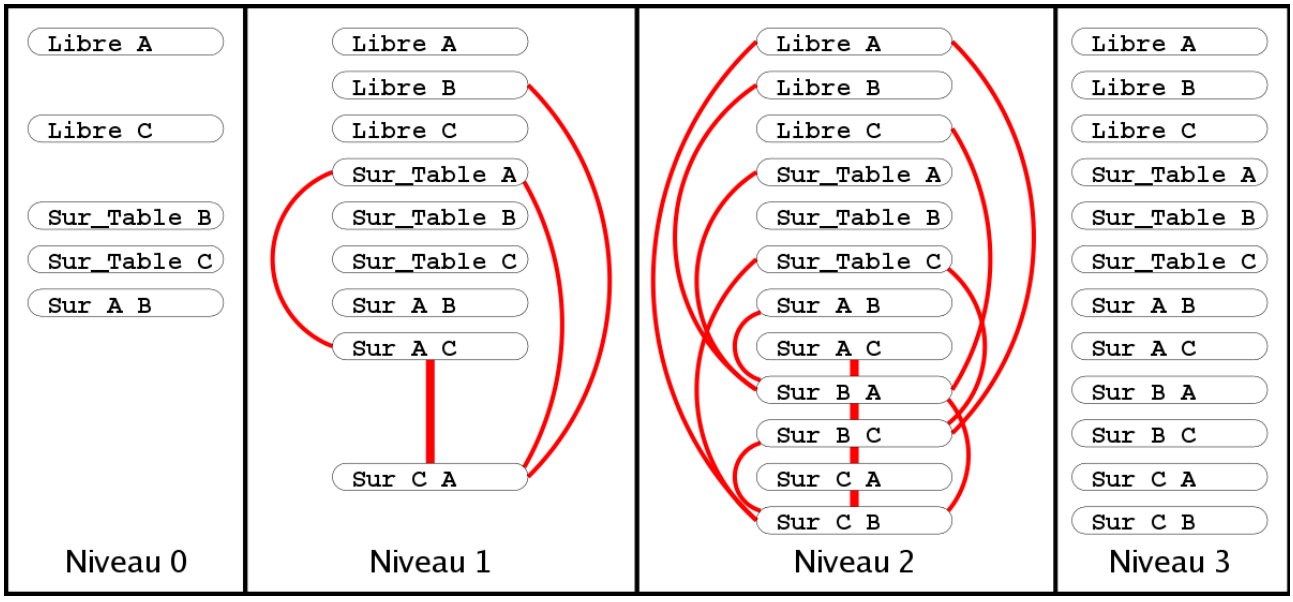


Figure 5.4: Construction du graphe d'accessibilité sur le problème des cubes (figure 5.3) avec création des « *mutex* » (les négations de faits ne sont pas représentées).

que les faits du but sont présents directement au deuxième niveau (figure 5.4). Or il faut au minimum trois actions pour trouver une solution. Le graphe d'accessibilité est donc un peu trop approximatif.

Pour limiter la perte d'information dûe à ce regroupement, les auteurs de GraphPlan, définissent la notion de *« mutex »* correspondant à une relation d'exclusion mutuelle. Deux faits seront dites *« mutex »* si et seulement s'il n'existe aucun état accessible depuis l'état initial dans lequel les deux faits soient vrais. A chaque niveau l'algorithme va calculer incrémentallement des *« mutex »* entre les actions et entre les faits :

- deux actions d'un niveau i sont mutex si :
  - les effets de l'une sont *« mutex »* avec les effets ou les préconditions de l'autre.
  - leurs préconditions sont *« mutex »* au niveau i.
- deux faits d'un niveau i sont *« mutex »* si :
  - l'un est la négation de l'autre.
  - toutes les actions de niveau i-1 supportant les faits sont *« mutex »* 2 à 2.

Une solution est trouvée au niveau n si tous les faits existent à ce niveau et qu'il n'y a pas de *« mutex »* entre eux. Dans l'exemple, nous voyons que tous les *« mutex »* ont disparu au niveau 3 (figure 5.4). Il se trouve effectivement que tous les états accessibles de l'exemple sont atteignables en 3 actions. Une propriété importante de ce graphe d'accessibilité est de fournir un nombre minimum d'action. En effet, si le but n'est pas présent dans le graphe d'accessibilité avant le niveau n, c'est qu'il n'existe aucun plan solution de moins de n actions.

Les planificateurs du type *« GraphPlan »* sont parmi les plus performants du domaine. Ils se sont fait dépasser récemment par des planificateurs ayant une bonne heuristique.

5.2.3 Avoir une bonne heuristique.

Classiquement, la planification dans l'espace d'état est effectuée par des algorithmes de recherche de chemin dans un graphe dont un des plus connus est A* [Nilson 80]. Cet algorithme permet de limiter la recherche en utilisant une heuristique qui représente la distance au but à atteindre.

Tout le problème vient du calcul de cette heuristique. Le calcul de la distance exacte est un problème plus complexe que la planification elle même. Pour être intéressante, cette heuristique doit être informative. En planification de mouvement il est possible d'utiliser la distance *« à vol d'oiseau »*, mais dans l'espace d'état cette distance n'est pas aussi évidente.

Il faut distinguer deux types d'heuristiques : les heuristiques calculées automatiquement, et celles fournies par l'utilisateur.

Dans le premier cas les planificateurs seront dit *« domaine indépendant »*. Les planificateurs *« HSP »* [Bonet 01] et *« FF »* [Hoffmann 01] en sont des exemples. Dans le second cas les planificateurs seront dit *« domaine dépendant »*. *« SHOP »* [D.S. Nau 99], *« TLPlan »* [Bacchus 96] et *« TalPlan »* [Doherty 99] font partie de cette catégorie.

Les planificateurs *« domaines dépendants »* sont nettement plus rapides que leurs homologues, par

contre pour chaque domaine rencontré une heuristique doit être fournie par l'utilisateur.

Pour *« SHOP »*, des méthodes de résolutions hiérarchiques dites *« HTN »* [Erol 95] sont données. Par exemple pour aller à l'aéroport vous pouvez soit prendre le bus soit le taxi. Pour prendre le bus, il faut aller à pied à la station et acheter un ticket. Pour prendre le taxi, il faut téléphoner. Ainsi la méthode *« aller à »* peut se diviser en deux sous-méthodes : prendre le bus ou prendre le taxi.

Pour *« TLPlan »* et *« TalPlan »* ce sont des règles de logique temporelle du premier ordre qui sont données. Par exemple dans le monde des cubes, il est possible de définir la notion de *« bonne tour »*, c'est une tour dont tous les cubes sont dans leur position but. Il est alors possible d'ajouter une règle qui indique qu'une *« bonne tour »* doit toujours rester bonne, c'est-à-dire que tout cube ajouté à cette tour doit être dans sa position finale.

Pour notre part nous avons choisi de nous orienter vers les planificateurs indépendants du domaine. Ce choix peut être discutable, vu que nous allons appliquer ce planificateur dans des contextes bien précis, à savoir les tâches de manipulation pour plusieurs robots [6]. Mais vu la simplicité des problèmes symbolique abordés il est beaucoup plus simple d'utiliser les planificateurs indépendants du domaine qui ne demande aucun ajout de connaissance de la part de l'utilisateur.

« FF » et *« HSP »* construisent leur propre heuristique. Ils sont basés sur le même principe. Puisqu'il est trop difficile de calculer la distance au but (longueur du plan) dans le domaine $\mathcal{D}$ donné, ils vont le faire dans un domaine relaxé $\mathcal{D}'$. La construction de ce domaine se fait en retirant des actions, tous les effets négatifs. Dans la version « `:strips` » de PDDL cela correspond à retirer tous les effets commençant par « `not` ». Les actions ne pourront plus supprimer de faits dans les états. Ainsi après une action *« allumer lampe »* les faits *« pièce sombre »* et *« pièce éclairée »* pourront être simultanément présents. Il sera alors plus facile de satisfaire les préconditions des actions et donc de trouver un plan solution.

« HSP » cherche un plan $\mathcal{P}_i$ indépendamment pour chaque fait demandé par le but. Son heuristique est alors égale à la somme des longueurs des plans $\mathcal{P}_i$ trouvés.

« FF » utilise un algorithme de type *« GraphPlan »* pour trouver une solution au domaine relaxée $\mathcal{D}'$. La longueur du plan ainsi obtenu sera son heuristique. L'avantage de cette méthode est de permettre de prendre en compte les interactions entre les faits du but.

Ainsi *« FF »* utilise un domaine relaxé $\mathcal{D}'$ pour appliquer une méthode qui va elle aussi simplifier le domaine $\mathcal{D}'$ avec un graphe d'accessibilité pour accélérer la recherche de solution. Comme nous le verrons par la suite, nous avons décidé d'utiliser *« Metric-FF »* [Hoffmann 02] pour calculer une heuristique pour aSyMov dans un domaine relaxé qui ne prendra pas en compte la géométrie de l'environnement. Cela rajoutera un nouveau niveau d'abstraction.

[6] Il serait en effet facile d'introduire des méthodes spécifiques à la planification de tâches de manipulation, par exemple pour déplacer un objet une méthode pourrait être : se déplacer pour saisir l'objet, prendre l'objet, le transporter à sa position but, le déposer.

5.2.4 Les limites des techniques de planification.

Des planificateurs indépendants du domaine ont été améliorés spectaculairement et peuvent résoudre des problèmes de plus en plus complexes.

Cependant, leur utilisation efficace en robotique est toujours très limitée. De notre point de vue, cela est dû principalement à la différence qu'il y a entre la représentation du monde pour les planificateurs de tâches et le monde réel. En effet ils ne prennent pas en compte les conséquences géométriques des actions. Par exemple, selon le contexte (taille et forme du robot et de l'objet, obstacle de l'environnement), un robot qui déplace un objet peut bloquer des passages utiles pour la suite du plan.

D'un autre côté les planificateurs de mouvement sont couramment utilisés en robotique, ils produisent des plans beaucoup plus réalistes qui peuvent être plus facilement appliqués sur les robots. Plusieurs tâches telles que : le transport coordonné d'objet, la manipulation, la coordination de mouvement, ont été développées. Mais la combinaison de ces tâches élémentaires afin d'accomplir une mission en prenant bien en compte les conséquences géométriques de chaque action est encore bien souvent réalisée au moyen de scripts (ou HTN) fournis par le programmeur.

Il semble alors intéressant de combiner ces deux approches afin de traiter les problèmes de manière générale. Il faut alors bien prendre en compte les spécificités de chacune des techniques.

La planification de tâches a besoin de travailler dans un espace d'état de taille limitée. La plupart des applications robotique qui utilisent des planificateurs de tâches avec des actions de déplacement, donnent un certain nombre de positions caractéristiques du problème. Il faut donner notamment : des positions intermédiaires pour qu'un robot ne gêne pas les déplacements des autres robots, des positions accessibles pour saisir un objet ...

Or nous voulons que le planificateur trouve lui-même ces positions. La planification de mouvement travaille dans un espace continu où une infinité de configurations est possible. Même si plusieurs d'entre elles ne seront pas utilisables car difficiles d'accès, le nombre de positions intéressantes peut-être très important.

Un autre point à prendre en compte est le caractère non déterministe de la planification de mouvements. Si deux robots peuvent déplacer un objet, le planificateur devra chercher une solution pour l'un, puis s'il n'y arrive pas, chercher une solution avec l'autre et recommencer à chercher alternativement avec l'un ou l'autre des robots. Comme il n'y a pas de critères permettant de dire que les tâches ne sont pas faisables, il n'y a pas moyen de stopper définitivement l'enrichissement des roadmaps. Ainsi l'apprentissage des roadmaps devra se faire en parallèle avec le système de recherche de solutions. Il pourra avoir création de positions intéressantes tout au long de la recherche.

Il faudrait disposer d'un planificateur de tâches capable d'accepter en cours de recherche une augmentation des faits de son état courant qui soit fonction des techniques d'apprentissage des roadmaps. Cet enrichissement de l'état serait complètement imprévisible puisque dépendant de techniques probabilistes. Nous avons décidé de ne pas aborder le problème de cette manière qui nous paraissait complexe, et qui remet en cause toutes les techniques de planification que nous avons vues précédemment.

Nous avons développé une autre façon d'aborder le problème. La planification de tâches se fait

sur un domaine abstrait du monde réel, elle travaille sur des symboles représentant des notions de l'environnement. Au lieu de représenter les configurations des robots au sein du planificateur de tâches nous avons choisi de représenter la fonction de ces configurations par des *« positions symboliques »*. Nous verrons cela un peu plus loin (§ 5.3.4).

De plus pour prendre en compte les contraintes géométriques sur des actions futures nous avons développé un mécanisme qui affine progressivement le choix des instances géométriques faites pour chaque *« positions symboliques »* (§ 6.3).

5.3 Les liens entre symbolique et géométrique.

ASyMov est un planificateur qui est destiné à travailler sur des problèmes qui mêlent à la fois des aspects géométriques et des notions symboliques. Pour cela nous allons décrire une sémantique que nous avons développée afin de lier ces deux modes de représentation.

Nous avons décidé qu'aSyMov serait un planificateur dans l'espace d'état. Il lui faut donc une heuristique pour guider sa recherche. Pour cela nous avons choisi de chercher un plan symbolique dans un problème relaxé où les contraintes géométriques ne seraient pas prises en compte. Nous avons intégré à aSyMov *« Metric-FF »* [Hoffmann 02], un planificateur *« domaine indépendant »* très performant, pour calculer cette heuristique.

Nous allons donc définir un domaine symbolique écrit en *« PDDL 2.1 »* (§ 5.1.3) utilisable par *« Metric-FF »* et compatible avec notre représentation géométrique. Sa sémantique sera élaborée de manière à rende compte entre autre de la composition des robots, des définitions des roadmaps et des liens entre roadmaps.

Comme nous voulons laisser une grande souplesse quant à la définition du domaine symbolique, nous n'allons pas exprimer des actions spécifiques au domaine géométrique. Nous avons plutôt exprimé ces relations à travers un certain nombre de types et de prédicats spécifiques qui vont correspondre à un problème *« basique »* de manipulation.

5.3.1 Les types spécifiques.

« PDDL 2.1 » permet de définir un certain nombre de types qui vont être spécifiques à aSyMov. Ils permettent de relier les objets, robots, roadmaps du domaine géométrique aux atomes du domaine symbolique.

Chaque atome pourra être relié à une ou plusieurs instances géométriques si celles-ci ont la même fonction symbolique.

Nous avons défini cinq types spécifiques à aSyMov :

- « `robot` » et « `object` » : ces deux types spécifient quand un atome est relié à un ou *plusieurs* robots et objets définis dans le domaine géométrique.

  En effet au niveau géométrique un robot transportant un objet peut avoir des cinématiques différentes suivant la définition de prise qu'il utilise. Cela peut mener à des définitions de robots

géométriques différents, mais qui feront un atome unique au niveau symbolique.

- « `symbolic-position` » : ce type permet de nommer des *« positions symboliques »* qui vont faire abstraction de l'instanciation précise des configurations dans le monde géométrique. Nous verrons plus en détail l'utilisation de ces positions après la définition des prédicats qui s'appliquent sur celles-ci (§ 5.3.4).
- « `roadmap-type` » : ce type permet de nommer un atome qui représentera une roadmap utilisée dans le domaine géométrique.

  Les roadmaps représentant les types de mouvements ou de placements possibles, ces atomes vont permettre de définir les types de mouvement autorisés et les types de placements des *« positions symboliques »*.

  Les atomes utilisés dans un problème *« basiques »* sont : *Transit* (`TI`), *Transfert* (`TA`), *Grasp* ∩ *Placement* (`GP`).

  Les atomes de type « `roadmap-type` » seront généralement des constantes du domaine.
- « `position-purpose` » : ce type permet de nommer des propriétés supplémentaires qui vont préciser certaines *« positions symboliques »*.

  Celles-ci ne seront définies que si elles ont une *« utilité »* symbolique au sein du problème.

  Cela peut-être la définition d'une configuration précise pour une position initiale ou finale, nous définirons alors les atomes `INIT` et `GOAL` qui pourront servir à préciser la fonction de certaines *« positions symboliques »*.

  Mais cela peut aussi être un atome `PHOTO` qui précisera que les *« positions symboliques »* associées permettent de prendre des photos.

  Les atomes de type « `position-purpose` » seront généralement des constantes du domaine.

Les atomes définis par ces cinq types seront reliés aux instances géométriques correspondantes à travers un fichier de liaison entre domaines (§ 6.2).

Un atome du type « `robot` » ou « `object` » sera associé à des chaînes cinématiques.

Les couples (« `robot` », « `roadmap-type` ») ou (« `object` », « `roadmap-type` ») seront associés à des couples (chaîne cinématique, roadmap associée). Les roadmaps associées devront être des roadmaps de mouvements ou des roadmaps affinables (§ 3.2).

Un atome « `position-purpose` » sera associée à des définitions de restriction de l'espace de configuration (par exemple configurations valides pour prendre une photo ou configuration initiale ou finale).

Un atome du type « `symbolic-position` » donnera un nom à une *« position symbolique »* qui sera associée à un ensemble de *« positions symboliques appliquées »*. Ces positions symboliques appliquées sont une structure propre à aSyMov qui vont représenter les configurations accessibles à un instant donné d'un plan (§ 5.3.4).

Nous avons choisi arbitrairement une certaine nomenclature pour ces atomes qui sera valable dans la suite de ce document. Un nom de *« position symbolique »* sera du type :

```
P_{robot}_{roadmap-type}_[fonction]
```

Où `fonction` précisera le rôle de la position. Par exemple « `P_R_TI` » représente une position quelconque du robot `R` dans une roadmap de type *Transit* (`TI`).

5.3.2 Les prédicats.

Nous distinguerons quatre types de prédicats en fonction de la relation qu'ils ont avec le fonctionnement d'aSyMov.

5.3.2.1 Les prédicats de définition des positions symboliques :

Ce sont des prédicats statiques (i.e. aucune action ne peut les modifier). Ils sont indispensables à aSyMov puisqu'ils décrivent les caractéristiques des positions symboliques. Une position symbolique représentera l'ensemble des nœuds de roadmaps qui correspondront aux critères donnés par ces prédicats.

- `(belongs-to ?p - symbolic-position`
  `?r - (either robot object)`
  `?type - roadmap-type) :`

La position « `?p` » est définie pour le robot (ou l'objet) « `?r` » et un type de roadmap « `?type` ». Par exemple `(belongs-to P_R_TI R TI)` pour une position quelconque du robot `R` appartenant à l'espace de *Transit*. Donc `P_R_TI` représentera l'ensemble des nœuds de la roadmap de *Transit* du robot `R`.

Il faut qu'il y ait un unique fait `(belongs-to P ?r ?t)` pour chaque position symbolique `P`.

L'ensemble des nœuds définit par le prédicat « `belongs-to` » pourra être contraint et réduit par les deux prédicats suivants.

- `(has-purpose ?p - symbolic-position`
  `?pos-fct - position-purpose) :`

La position « `?p` » a une propriété spécifique décrite par « `?pos-fct` ». Elle va avoir un traitement spécial comme par exemple une position initiale `(has-purpose P_R_TI_INIT INIT)`, finale `(has-purpose P_R_TI_GOAL GOAL)`, ou une position ayant une signification symbolique comme les configurations permettant de prendre de la colle dans un conteneur `(has-purpose P_R_TI_PREND_COLLE PREND_COLLE)`. La position `P_R_TI_INIT` sera représentée par une configuration unique, donc par un unique nœud.

- `(connection ?p1 - symbolic-position`
  `?p2 - symbolic-position) :`

Ce prédicat permet d'indiquer des liens entre roadmaps. Par exemple :

```
(belongs-to P_R_TI_R-O_GP R TI)
(belongs-to P_R-O_GP R-O GP)
```

`(connection P_R_TI_R-O_GP P_R-O_GP)`

Les prédicats « `belongs-to` » indiquent que `P_R_TI_R-O_GP` représentent l'ensemble des nœuds de la roadmap de *Transit* du robot `R` et que `P_R-O_GP` représente l'ensemble des nœuds de la roadmap de *Grasp* ∩ *Placement* du robot composé `R-O` (robot `R` saisissant l'objet `O`).

« `connection` » restreint la définition de `P_R_TI_R-O_GP` pour ne prendre en compte que les nœuds qui sont liés à des nœuds de la roadmap de *Grasp* ∩ *Placement* du robot composé `O-R`.

Un prédicat « `connection` » doit donc correspondre à un *« lien d'héritage »* (§ 3.3) défini au niveau géométrique.

`P_R_TI_R-O_GP` va donc représenter des positions de *Transit* du robot `R` qui permettent de saisir l'objet `O`.

5.3.2.2 Le prédicat de positionnement des objets et des robots :

C'est un prédicat dynamique (modifiable par les actions) propre à aSyMov. Il est défini par :

```
(on  ?r - (either robot object)
     ?p - symbolic-position) :
```

Il signifie que le robot ou l'objet « `?r` » est à la position symbolique « `?p` ». Ainsi une action de déplacement ou de changement de roadmap pour un même robot aura des effets du type :

`:effets (and (not (on R P1)) (on R P2))`

Vu la définition que nous avons prise pour définir les robots il aura aussi un sens *« d'existence »*. Quand un robot `R` saisit un objet `O` nous avons dit qu'il formait un nouveau robot `R-O`. Ainsi le robot `R` n'existera plus en tant qu'entité isolée. Une action de prise aura un effet du type :

`:effets (and (not (on R P1)) (not (on O P2)) (on R-O P3))`

Notons que `P3` doit être liée à `P1` et `P2` par des prédicats « `connection` » afin de s'assurer que ce changement se fait pour une même instance géométrique. Le robot composé n'a pas bougé dans l'opération de saisie.

Les prédicats « `on` » vont représenter les seuls faits dynamiques de l'état symbolique qui auront des conséquences géométriques.

5.3.2.3 Les prédicats de notions géométriques :

Ces prédicats vont représenter des notions géométriques au sein de la planification de tâches symboliques. Ils vont servir à s'assurer que les actions symboliques sont bien conformes à la géométrie du problème et parfois aider la recherche.

ASyMov qui connaît à la fois les descriptions géométrique et symbolique du problème ne va pas se servir directement de ces prédicats. Ainsi la typographie des prédicats qui vont représenter ces notions, est libre. De plus aSyMov ne suppose pas que ces prédicats soient statiques. Par contre ces notions sont fondamentales pour décrire des actions valides géométriquement.

• La première notion introduite est la composition des robots. Il faut que les actions décrites au niveau symbolique respectent cette composition. Pour cela nous pouvons utiliser le prédicat suivant :

```
(composed ?r1 - (either robot object)
          ?r2 - (either robot object)
          ?rtot - (either robot object))
```

Cela signifie que deux robots (ou objets) `?r1` et `?r2` peuvent s'assembler pour en former un troisième `?rtot`. Par exemple, un robot `R` peut saisir un objet `O` et former le robot `R-O` : `(composed R O R-O)`.

« `composed` » indique la possibilité d'une composition, `R-O` n'existe que s'il y a une instance du prédicat « `on` » `(on R-O ?p)`.

Il est possible aussi d'avoir des tâches d'assemblage, par exemple un pied de table `PIED` peut se fixer à un plateau `PLATEAU` pour former une table `TABLE` : `(composed PIED PLATEAU TABLE)`.

Dans le cas de saisie simultanée il est possible d'utiliser d'autres prédicats tel que « `compose3` » qui associerait simultanément 3 objets, pourvu que cette composition ait aussi été définie au niveau géométrique.

• La seconde notion est facultative, mais elle peut aider grandement le planificateur. C'est la notion de connexité entre positions symboliques. Par défaut aSyMov considère que les positions symboliques qui ont le même prédicat « `belongs-to` » (même robot, même type de roadmap) sont connexes. Ceci peut être restreint pour aider le planificateur. Pour cela il est possible d'utiliser le prédicat :

```
(path ?p1 - symbolic-position
      ?p2 - symbolic-position)
```

Cela peut permettre d'inclure une connaissance a priori de la topologie du domaine. Par exemple si nous avons des positions correspondant à deux pièces « `P_R_TI_PIECE_1` » et « `P_R_TI_PIECE_2` » et une machine à café dans la seconde pièce « `P_R_TI_CAFE` » alors nous pouvons restreindre la recherche avec le prédicat « `path` » : `(path P_R_TI_PIECE_2 P_R_TI_CAFE)` et `(not (path P_R_TI_PIECE_1 P_R_TI_CAFE))`. Bien entendu aSyMov peut trouver cette connexité au niveau géométrique, mais cette indication peut aider la recherche d'une façon non négligeable.

Nous l'avons utilisé principalement sur un domaine avec une porte que le robot devait ouvrir. Cela nous a permis d'indiquer au niveau symbolique que pour passer d'une pièce à l'autre il fallait ouvrir la porte.

5.3.2.4 Les prédicats supplémentaires :

Ces prédicats vont représenter des notions qui ne sont pas géométriques telles que le niveau d'une batterie, le fait qu'un objet est comestible, ... Ils ne font pas partie d'un problème « basique » de manipulation, mais ils vont permettre d'enrichir la représentation du monde.

Ils peuvent avoir une relation indirecte avec la géométrie. Par exemple un prédicat comme « `glue-tank` » qui représente la quantité de colle qu'a le robot ne peut être modifié qu'à certaines positions que ce soit pour le remplir ou pour utiliser cette colle.

L'ensemble de ces prédicats permettent à l'utilisateur de définir une très grande variété de problèmes tout en fixant un certain formalisme pour la représentation des actions prenant en compte directement la géométrie.

5.3.3 Exemple d'actions.

Après avoir présenté les types et les prédicats utilisés dans le planificateur de tâches, nous allons présenter des exemples d'actions couramment utilisées et des remarques propres à cette représentation.

Une action de « `goto` » peut être utilisée pour symboliser des mouvements, c'est-à-dire l'utilisation de méthodes locales définies dans les roadmaps, et donc des mouvements au sein d'une même composante connexe. Elle peut être écrite par :

```
(goto
  :parameters (?r - robot ?p_r_1 - symbolic-position
               ?p_r_2 - symbolic-position ?t - roadmap-type)
  :precondition (and (belongs-to ?p_r_1 ?r ?t)
                     (belongs-to ?p_r_2 ?r ?t)
                     (path ?p1 ?p_r_2)
                     (on ?r ?p_r_1))
  :effect (and (not (on ?r ?p_r_1)) (on ?r ?p_r_2))
)
```

Une autre action couramment utilisée est l'action de « `grasp` » d'objet :

```
(grasp
  :parameters (?r  - robot ?p_r  - symbolic-position
               ?o  - object ?p_o  - symbolic-position
               ?r-o - robot ?p_r-o - symbolic-position
               ?t  - roadmap-type)
  :precondition (and (composed ?r ?o ?r-o)
                     (connection ?p_r ?p_r-o)
                     (connection ?p_o ?p_r-o)
                     (belongs-to ?p_r-o ?r-o ?t)
                     (on ?r ?p_r) (on ?o ?p_o))
  :effect (and (not (on ?r ?p_r)) (not (on ?o ?p_o))
               (on ?ro ?p_r-o))
)
```

Remarque :
Le prédicat « `composed` » est indispensable pour s'assurer que l'on a bien les mêmes chaînes cinématiques élémentaires avant et après l'action.
L'action opposée à « `grasp` » est « `ungrasp` » et se définit de manière symétrique.

L'action de changement de type de roadmap peut aussi se faire pour un même robot quand par exemple le robot passe de *Grasp* ∩ *Placement* à des mouvements dans *Transfert*. Cela peut être représenté par l'action suivante :

```
(switch
  :parameters (?r  - robot ?p_r_1  - symbolic-position
                           ?p_r_2  - symbolic-position
               ?t2  - roadmap-type)
  :precondition (and (connection ?p_r_1 ?p_r_2)
                     (belongs-to ?p_r_2 ?r ?t2)
                     (on ?r ?p_r_1))
  :effect (and (not (on ?r ?p_r_1))
               (on ?r ?p_r_2))
)
```

Les actions précédentes sont caractéristiques d'un problème *« basique »* de manipulation.

Il est tout à fait possible d'avoir d'autres actions qui vont mélanger les aspects géométriques aux aspects purement symboliques, par exemple pour prendre de la colle :

```
(get-glue
  :parameters (?r - robot ?p_r - symbolic-position
               ?rsym - type-robot-symbolique)
  :precondition(and (is-robot-sym ?r ?rsym)
                    (> (tank-capacity ?rsym)
                       (glue-tank ?rsym))
                    (has-purpose ?p_r GET_GLUE)
                    (on ?r ?p_r))
  :effect (increase (glue-tank ?rsym) 1)
)
```

Remarque :
Dans cet exemple `(glue-tank ?rsym)` et `(tank-capacity ?rsym)` sont des variables numériques (gérées par *« Metric-FF »*) représentant respectivement la capacité actuelle du réservoir et sa contenance maximale.

Le prédicat `(is-robot-sym ?r ?rsym)` et le type « `type-robot-symbolique` » permettent de faire partager des caractéristiques communes aux robots composés `R-O`$_i$ issus d'un même robot `R`. En effet, un robot ne perd pas sa colle quand il prend un objet, donc le robot composé et le robot seul héritent la même ressource de `?rsym`.

Ce dernier exemple montre qu'il est possible de créer un grand nombre d'actions qui vont faire interagir les contraintes géométriques. Même les actions du problème *« basique »* de la manipulation peuvent être redéfinies. Cette souplesse provient du fait que nous avons choisi de lier le symbolique et le géométrique au niveau des prédicats, alors que classiquement ce lien ce fait au niveau des actions.

5.3.4 Les positions symboliques.

Le monde réel est continu. Or les techniques de planification de tâches utilisent une représentation discrète du monde et il est difficile d'enrichir cette représentation au cours de la planification. Nous avons donc choisi d'avoir pour un domaine donné, un nombre fixe de positions pour les robots et les objets. Ainsi nous ne pouvons pas associer une position symbolique à un nœud d'une roadmap puisque le nombre de nœuds va évoluer au cours du temps. L'idée principale de notre représentation est de représenter une position par sa fonction. Cette position sera appelée *« position symbolique »*, et pourra avoir une infinité d'instances géométriques possibles, ou plus précisément un nombre indéfini à priori de nœuds de roadmaps.

Une *« position symbolique »* est décrite par les prédicats « `belongs-to` », « `has-purpose` » et « `connection` » (§ 5.3.2) qui définissent le type de mouvements autorisé (*Transit*, *Transfert*, ...) et la fonction de la position (position pour prendre une photo, pour saisir un objet, ...). A chaque position symbolique `P`$_i$ on peut donc associer un ensemble Ω_i de nœuds de roadmaps qui satisfont les conditions imposées par ces prédicats.

Ainsi `(belongs-to P_O_TI_INIT TI)` définit l'ensemble des nœuds de l'objet `O` de sa roadmap de *Placement*. `(connection P_O_TI_INIT P_R-O_GP)` restreint cet ensemble aux nœuds qui sont reliés à la roadmap de *Grasp* $\cap$ *Placement* du robot composé `R-O`. `(« has-purpose » P_O_TI_INIT INIT)` restreint encore cet ensemble au nœud ayant la configuration de la position initial de l'objet. Dans ce cas Ω contient au plus un nœud correspondant à la configuration de départ de l'objet.

A chaque fois qu'une position symbolique `P`$_i$ sera utilisée pour la création d'un nouvel état, c'est-à-dire, à chaque fois qu'une action ajoutera un prédicat `(on ?r P`$_i$`)` une *« position symbolique appliquée »* `P`$_i$`#k` associée à cette position symbolique sera créée.

Une *« position symbolique appliquée »* définit une position symbolique `P`$_i$ à un instant donné du plan. Elle représente le sous-ensemble des nœuds de Ω_i qui sont accessibles à ce niveau du plan. Elle est donc définie en fonction des positions symboliques appliquées précédentes.

Par exemple si deux robots peuvent transporter un objet `O` et qu'ils utilisent une action pour le poser en `P_O_TI`. Il n'y a aucune raison pour que les roadmaps de *Grasp* $\cap$ *Placement* des deux robots

soient identiques, donc aucune raison que les nœuds sur lesquels peut se trouver l'objet `O` déposé par un robot soient les mêmes que s'il est posé par l'autre robot, nous aurons donc deux positions symboliques appliquées `P_O_TI#1` et `P_O_TI#2`.

Ce sous-ensemble de nœuds Acc_i de Ω_i sera appelé liste d'accessibilité de la position symbolique appliquée (figure 5.6).

Une *« position symbolique appliquée »* définira cette liste d'accessibilité à travers des liens qu'elle aura avec les positions symboliques appliquées précédentes (qui ont été utilisées pour la créer) et suivantes (qui se servent d'elle).

La recherche de ces liens se fait à travers les effets sur les faits « `on` » des actions utilisées. Les faits « `on` » supprimés représentent les positions précédentes et les faits « `on` » ajoutés les suivantes. S'il existe des liens entre les positions symboliques précédentes et suivantes (« `belongs-to` » portant sur le même robot et la même roadmap ou « `connection` ») alors il faut créer un lien au niveau des positions symboliques appliquées.

Pour les actions définies plus haut, cela revient à :

- Pour « `goto` » : crée un lien entre « `?p_r_1` » et « `?p_r_2` » car elles appartiennent à la même roadmap.
- Pour « `grasp` » : crée un lien entre le couple (« `?p_r` », « `?p_o` ») et « `?p_r-o` » car elles sont connectées les unes au autres.
- Pour « `prend-colle` » : ne rien faire au niveau géométrique (pas d'effets sur le prédicat « `on` »).

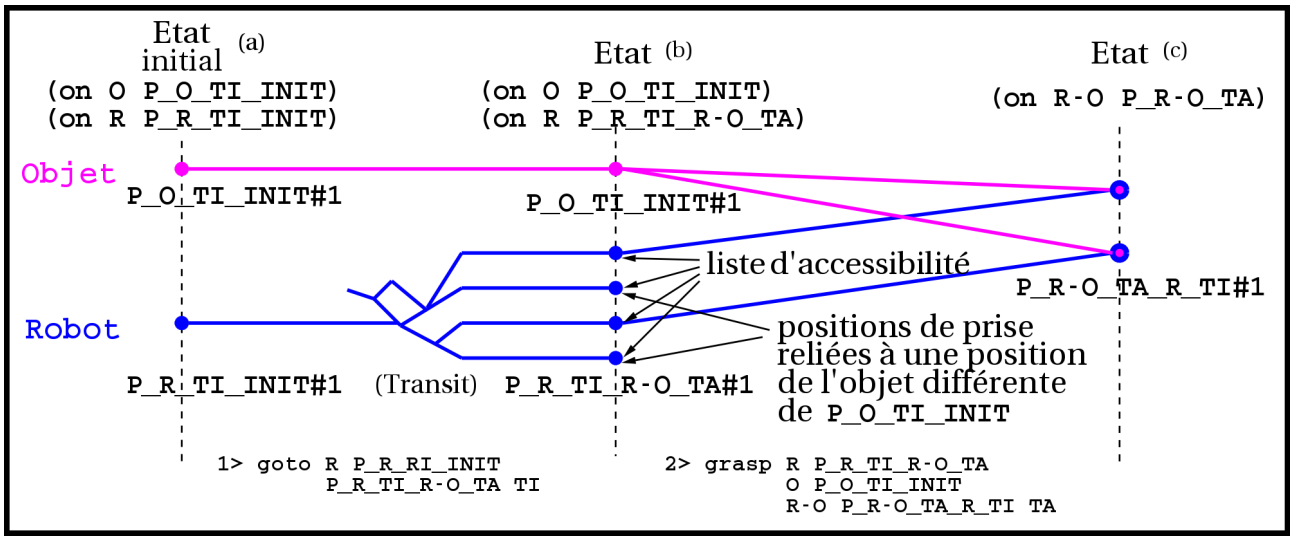


Figure 5.5: Liens entre les éléments des listes d'accessibilités pour les deux premières actions de l'exemple de la figure 5.6.

La liste d'accessibilité Acc_i de la *« positions symboliques appliquées »* est définie en fonction des liens que celle-ci a avec ses positions symboliques appliquées précédentes. Il existe alors trois possibilités :

- La position symbolique appliquée n'a pas de précédent (position initiale ou création d'objets

dans le monde). Dans ce cas les éléments de Acc_i n'ont pas de précédent et Acc_i est égale à Ω_i. (figure 5.5.a `P_R_TI_INIT` et `P_O_TI_INIT`).

- La position symbolique appliquée a un lien sur une position précédente P_j avec un fait « `belongs-to` » identique. Les deux positions font donc partie de la même roadmap et les nœuds sont liés par des composantes connexes (figure 5.5.b `P_R_TI_R-O_TA`). Les éléments de Acc_i sont les nœuds des composantes connexes des éléments de Acc_j. Cela correspond à une action de déplacement.
- La position symbolique appliquée possède une ou plusieurs positions précédentes (P_j, ..., P_k) avec des faits « `belongs-to` » différents. Les liens sont alors des liens entre roadmaps (figure 5.5.c `P_R_TI_R-O_TA`). Les éléments précédents de Acc_i sont les couples d'éléments de (Acc_j, ..., Acc_k) définis par ces liens entre roadmaps. Cela correspond à des actions de changement de roadmaps.

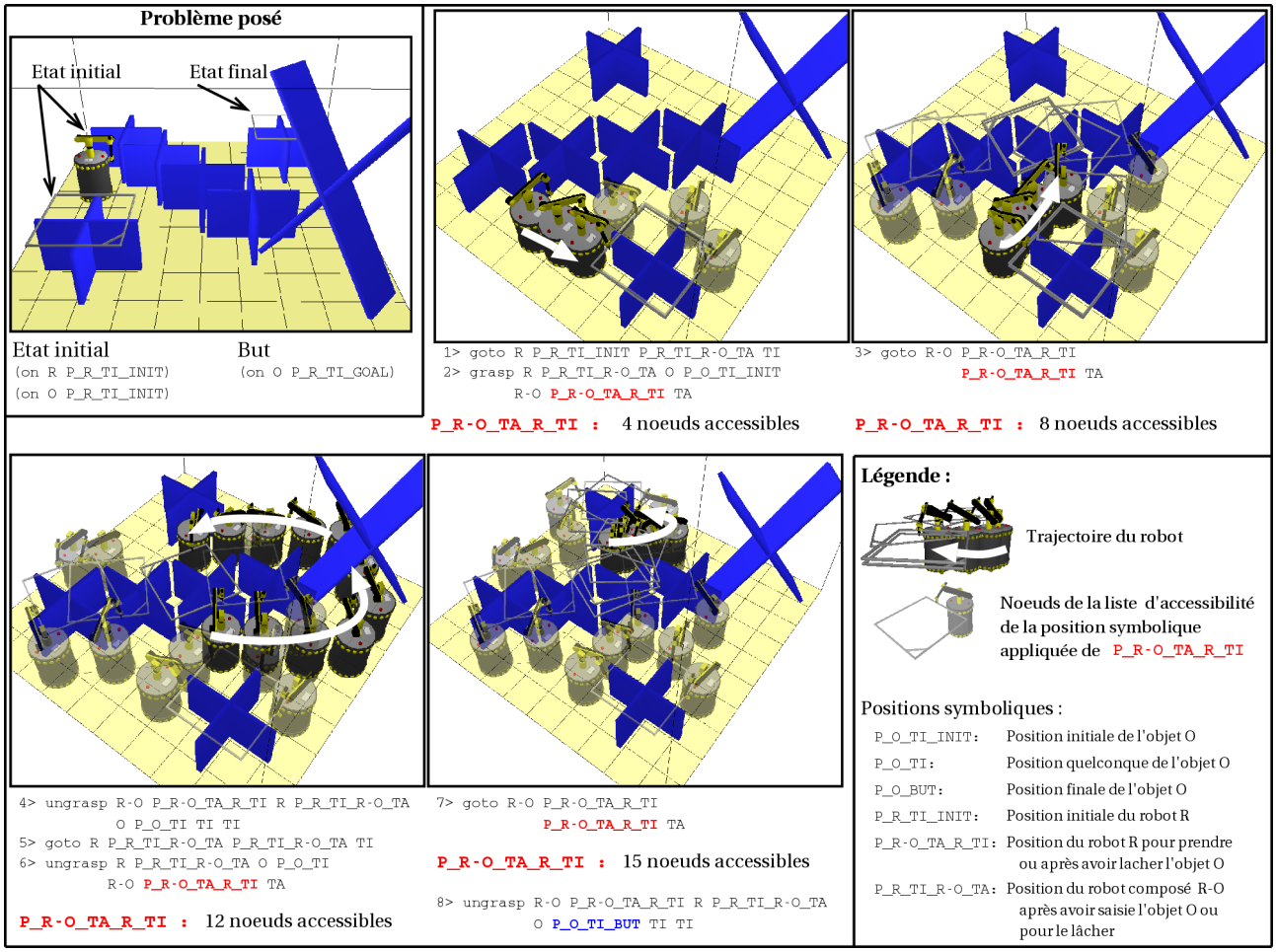


Figure 5.6: Positions symboliques appliquées sur un problème de manipulation avec un robot et un objet. Le robot ne pouvant pas passer sous le grand X avec l'objet. Nous pouvons remarquer l'augmentation du nombre de nœuds des listes d'accessibilités (robot transparent) correspondant à une même position symbolique `P_R-O_TA_R_TI`.

Les positions symboliques ainsi définies vont pouvoir constituer une abstraction du problème géométrique. Leur nombre sera fixe pour un problème donné. Il se peut qu'une position symbolique ait au cours du plan des instances possibles différentes. La figure 5.6 montre comment une même position symbolique `P_R-O_TA_R_TI`) va avoir des positions symboliques appliquées avec des listes d'accessibilités qui vont croître jusqu'à ce qu'elles puissent contenir le but.

Ces listes d'accessibilités vont jouer un rôle important dans la définition de l'espace d'état (§ 6.1) et dans la fonction de validation des positions (§ 6.3).

5.3.5 Remarques et conclusions.

Nous avons présenté un ensemble de types et de prédicats spécifiques que nous devons utiliser dans aSyMov pour pouvoir lier la représentation symbolique à la représentation continue du monde. Ces prédicats permettent de définir une abstraction du problème géométrique, ainsi un robot au niveau symbolique pourra représenter un ensemble de chaînes cinématiques différentes en fonction des positions de prise envisagées. De même une position symbolique ne représentera pas à proprement parler une position, mais les caractéristiques de cette position.

Au niveau des actions nous pouvons remarquer certaines différences par rapport au planificateur de tâches classique. Comme le montre la figure 5.6 l'action 3 `(goto P_R-O_TA_TI P_R-O_TA_TI)` fait un déplacement entre deux positions symboliques identiques. Par contre les positions symboliques appliquées sont différentes. Nous avons en fait `(goto P_R-O_TA_TI#1 P_R-O_TA_TI#2)`. Nous autorisons de tels déplacement quand ils permettent d'augmenter les listes d'accessibilités des positions symboliques appliquées.

La figure 5.5 montre que les actions de saisie (action 2), ont tendance à réduire les listes d'accessibilité car toutes les combinaisons ne sont pas forcément possibles. Ainsi la liste d'accessibilités de `P_R_TI_TA` qui représentaient toutes les positions où le robot pouvait saisir l'objet (action 1) est réduite pour ne contenir que les positions où il peut saisir l'objet dans sa position initiale (action 2).

L'utilisation de *« Metric-FF »* (§ 6.2) pourrait donc être remis en cause pour pouvoir ajouter au sein du planificateur de tâches des connaissances propres à la manipulation.

Dans le chapitre suivant nous allons enfin décrire l'algorithme complet du planificateur *« aSyMov »*.

Chapitre 6

L'algorithmique d'aSyMov.

Ce chapitre présente le planificateur élaboré pendant cette thèse : *« aSyMov »* [1]. Les chapitres précédents nous ont permis de définir les outils géométrique et symbolique qui seront utilisés.

ASyMov est un planificateur dans l'espace d'état, mais afin de diminuer la taille de cet espace il va regrouper les positions en classes d'équivalences appelées *« positions symboliques appliquées »* (§ 5.3.4). Il va en résulter une définition des états d'aSyMov (§ 6.1) qui possédera plusieurs niveau d'abstraction.

ASyMov est un planificateur qui commence sa recherche à partir de l'état initial (§ 6.2.1) car il doit être capable de connaître la position de tous les robots et les objets pour valider les actions géométriques. Il est guidé par une heuristique qui prend en compte à la fois les aspects symboliques et géométriques (§ 6.5).

Même si aSyMov fait une recherche *« en avant »*, il est aussi capable de changer les choix des instanciations géométriques de ses états pour satisfaire les contraintes d'un nouvel état. Le processus de validation effectue alors une propagation arrière des contraintes géométriques (§ 6.3.2).

De plus aSyMov est aussi un planificateur géométrique probabiliste, nous lui avons donc ajouté des *« actions d'apprentissages »* pour capturer la topologie du problème (§ 6.4.1).

Enfin comme les plans d'aSyMov ne sont pas optimaux (au moins du point de vue géométrique), nous avons ajouté à aSyMov une opération de post-traitement permettant de les optimiser et de les paralléliser (§ 6.6).

Nous allons successivement détailler au long de ce chapitre l'espace d'état d'aSyMov, son algorithme général, son processus de validation des actions et son apprentissage de la topologie, puis nous verrons le calcul de son heuristique et terminerons par les post-traitements qui sont appliqués sur un plan solution.

[1] a Symbolic Move3D

6.1 L'espace de recherche d'aSyMov.

ASyMov est un planificateur qui fait une recherche en avant dans l'espace d'état. Nous avons choisi ce type de planificateur pour les raisons suivantes :

- Performance des planificateurs de tâches de ce type grâce à l'utilisation de *bonnes* heuristiques.
- A chaque étape de la recherche nous disposons d'un état complet, ce qui nous permet de valider les chemins ou d'enrichir les roadmaps en prenant en compte tous les corps mobiles de l'environnement.

Toutefois, pour ne pas avoir un espace de recherche trop grand, nous avons choisi de regrouper dans un même état l'ensemble de ses instanciations géométriques possibles. Pour cela un *« état d'aSyMov »* se décomposera en trois parties représentant différents niveaux d'abstraction. Il sera défini par :

$$\mathcal{E} = (\mathcal{E}_S, \mathcal{E}_A, \{\mathcal{E}_{G_1}, \ldots, \mathcal{E}_{G_n}\})$$

$\mathcal{E}_S$ représentera la partie symbolique du problème, il sera appelé : *« état symbolique »*. $\mathcal{E}_A$ représentera les positions accessibles, il sera appelé : *« état d'accessibilité »*. $\mathcal{E}_{G_i}$ représentera une instanciation géométrique de $\mathcal{E}_S$, il sera appelé : *« état géométrique »*. Le tout formera un *« état d'aSyMov »*.

6.1.1 L'état symbolique.

L'état symbolique rassemblera les faits de l'état tel qu'ils sont décrits dans le chapitre précédent (§ 5.3). Il permet notamment de représenter des notions qui n'apparaissent pas au niveau géométrique.

Par exemple pour un état initial nous pouvons avoir défini toutes les positions, mais dans un cas le robot aura une réserve de colle nulle, dans l'autre elle sera pleine. La seule différence proviendra alors de la variable d'état `(glue-tank R)` de *« l'état symbolique »*.

Nous avons dit que aSyMov allait utiliser *« Metric-FF »* pour calculer un plan heuristique. Les *« états symboliques »* correspondent exactement aux états utilisés par *« Metric-FF »*.

6.1.2 L'état d'accessibilité.

Cet état correspond à l'ensemble des *positions symboliques appliquées* correspondant aux *positions symboliques* actives de *l'état symbolique*, c'est à dire les *positions symboliques* des faits « `on` ».

$\mathcal{E}_A = (Acc_1, \ldots, Acc_n)$ avec Acc_i liste d'accessibilité de $\texttt{P}_i$ telle que `(on` $\texttt{R}_i$ $\texttt{P}_i$`)` $\in \mathcal{E}_S$.

Les listes d'accessibilités des positions symboliques appliquées permettent de définir les nœuds possibles pour chaque position.

La figure 6.1 montre que pour un même état symbolique, l'état d'accessibilité peut être différent. L'état d'accessibilité représente non seulement l'état courant, mais aussi toutes les positions possibles permises par le plan qui a mené à cet état.

Par contre *« l'état d'accessibilité »* ne prend pas en compte les collisions qu'il peut y avoir entre robots sur ces positions et cela tout au long du plan qui a mené à cet état. Donc *« l'état d'accessibilité »* résulte d'un problème relaxé. Il ne garantit pas l'existence d'une solution.

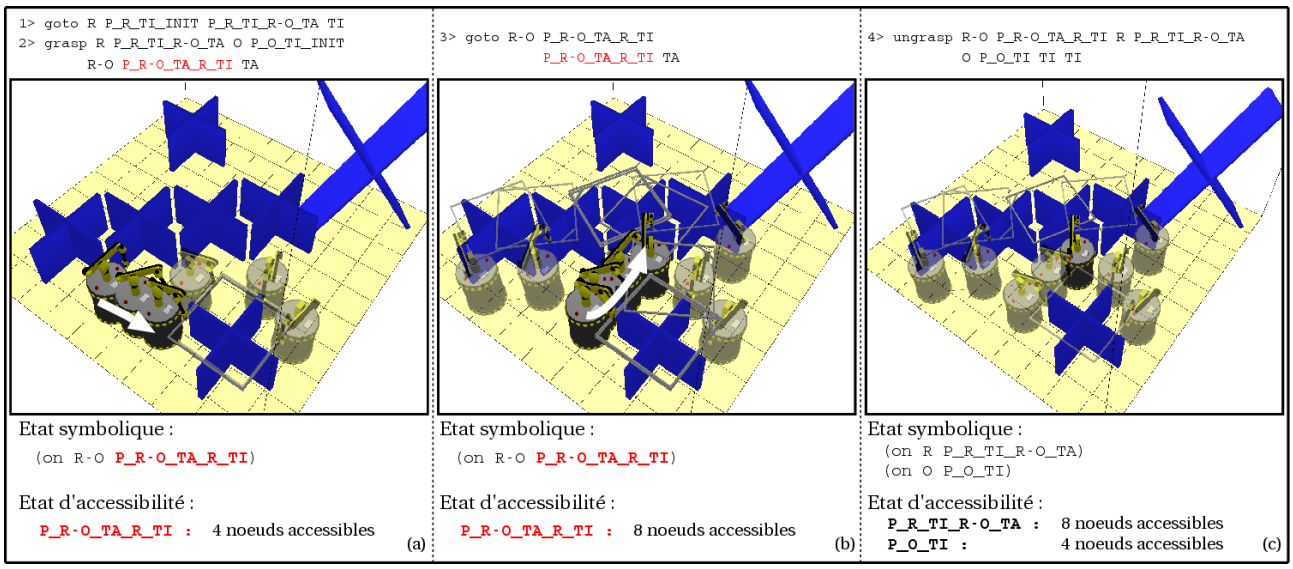


Figure 6.1: État d'accessibilités sur un problème de manipulation avec un robot et un objet (figure 5.6). Pour un même état symbolique (a) et (b) l'état d'accessibilité (robots transparents) peut être différent.

6.1.3 L'état géométrique.

« L'état géométrique » représente une instance géométrique de *l'état symbolique*, c'est l'attribution d'une configuration (d'un noeud) pour chaque *position symbolique* active. Un *« état géométrique »* est défini par :

$$\mathcal{E}_{G_i} = (N_1, \ldots, N_n) \text{ avec } N_i \in Acc_i \text{ et } Acc_i \in \mathcal{E}_A.$$

Le couple $(\mathcal{E}_S, \mathcal{E}_{G_i})$ représente un état au sens classique du terme, mais pour diminuer la complexité de l'état nous avons regroupé les instances géométriques. Un *« état d'aSyMov »* possède donc un ensemble *« d'état géométrique »*.

L'existence d'un *« état géométrique »* au sein d'un *état d'aSyMov* assure qu'il existe un ensemble de trajectoires valides, correspondant aux actions du plan symbolique qui a mené à la création de cet état, et qui permet d'atteindre les configurations de l'état géométrique.

Par extension, nous disons qu'un *« état d'aSyMov »* est *« valide »* s'il a au moins un *« état géométrique »*.

L'état d'accessibilité permet d'avoir l'ensemble des états géométriques possibles, mais la relaxation de la contrainte de non collision peut conduire à des *états d'accessibilité* identiques pour des états d'aSyMov différents. La figure 6.2 montre un tel exemple. Suivant l'ordre des actions, les états géométriques atteignables peuvent être différents. Les *états symboliques* et les *états d'accessibilités* sont identiques en (a) et (b), mais ces 2 états d'aSyMov sont différents car il y a des *états géométriques* qui sont différents entre (a) et (b).

Soit $\mathcal{GE}$ l'ensemble des états géométriques valides possibles pour un état d'aSyMov $\mathcal{E}$, alors :

$$\{\mathcal{E}_{G_1}, \ldots \mathcal{E}_{G_k}\} \subset \mathcal{GE} \subset Acc_1 \times \cdots \times Acc_n \text{ avec } \mathcal{E} = (\mathcal{E}_S, \mathcal{E}_A = (Acc_1, \ldots, Acc_n), \{\mathcal{E}_{G_1}, \ldots, \mathcal{E}_{G_k}\}).$$

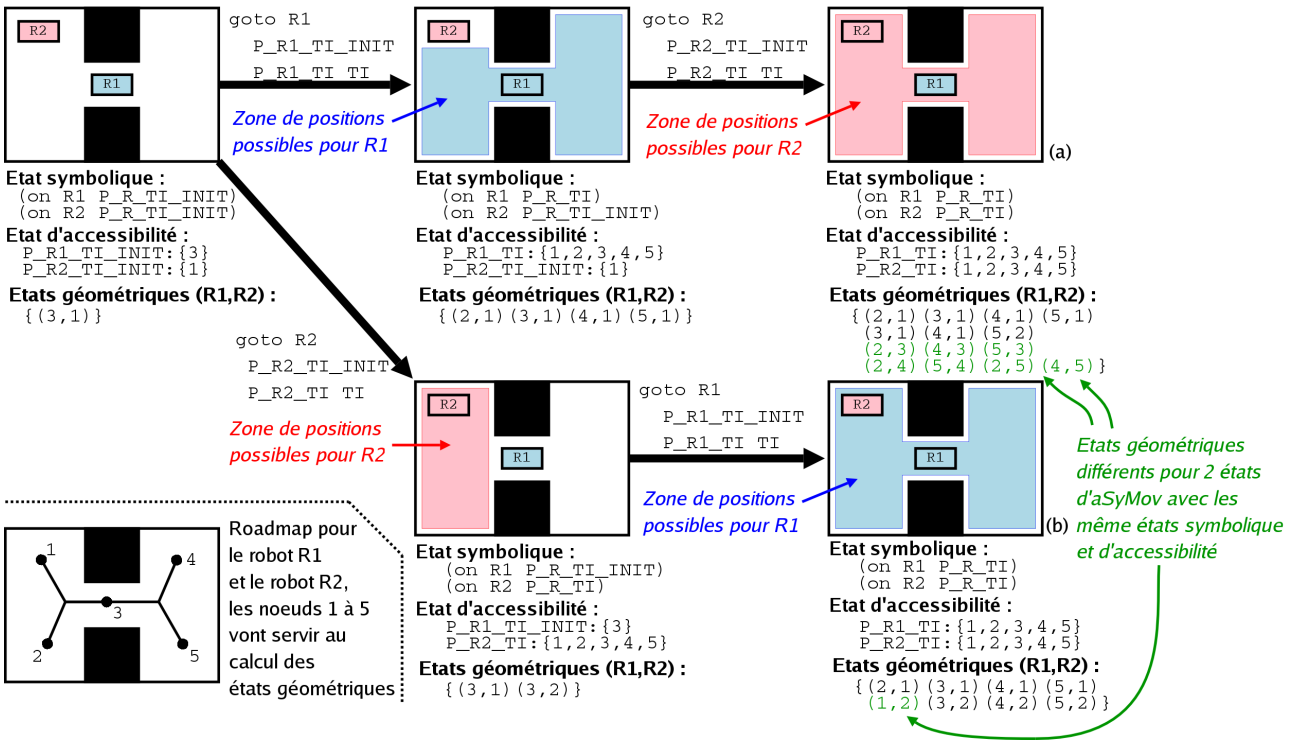


Figure 6.2: État d'aSyMov avec deux robots. (a) et (b) montre que l'ordre des actions est important pour définir les états géométriques possibles.

Il est trop complexe d'explorer entièrement le produit cartésien des listes d'accessibilités de $\mathcal{E}_A$ pour crée $\mathcal{GE}$. En effet il faudrait vérifier qu'il existe un plan géométriquement valide pour atteindre les *états géométriques* formés par ce produit cartésien des listes d'accessibilités (exponentiel en nombre de robots). Dans l'exemple trivial de la figure 6.2 les listes d'accessibilités ont 5 éléments chacune ce qui fait 25 possibilités. Parmi celles-ci les états d'aSyMov (a) et (b) possèdent respectivement 14 et 8 états géométriques valides possibles.

Puisque le nombre de combinaisons est trop grand, aSyMov va chercher à valider un minimum *d'états géométriques* nécessaires. Nous verrons plus en détail ce point dans la section 6.3 qui décrira le processus de validation des actions.

Remarque :
Les problèmes qui font apparaître des différences au niveau des *états géométriques* qui ne sont pas apparentes sur le reste de l'état d'aSyMov sont les problèmes les plus complexes. Dans ce cas, ni l'heuristique géométrique (basée sur une notion de connexité à travers les GCEs) ni l'heuristique symbolique ne peuvent guider la recherche.

6.2 L'architecture d'aSyMov.

Afin de résoudre un problème, aSyMov utilise trois bases de données pour décrire le problème (figure 6.3) :

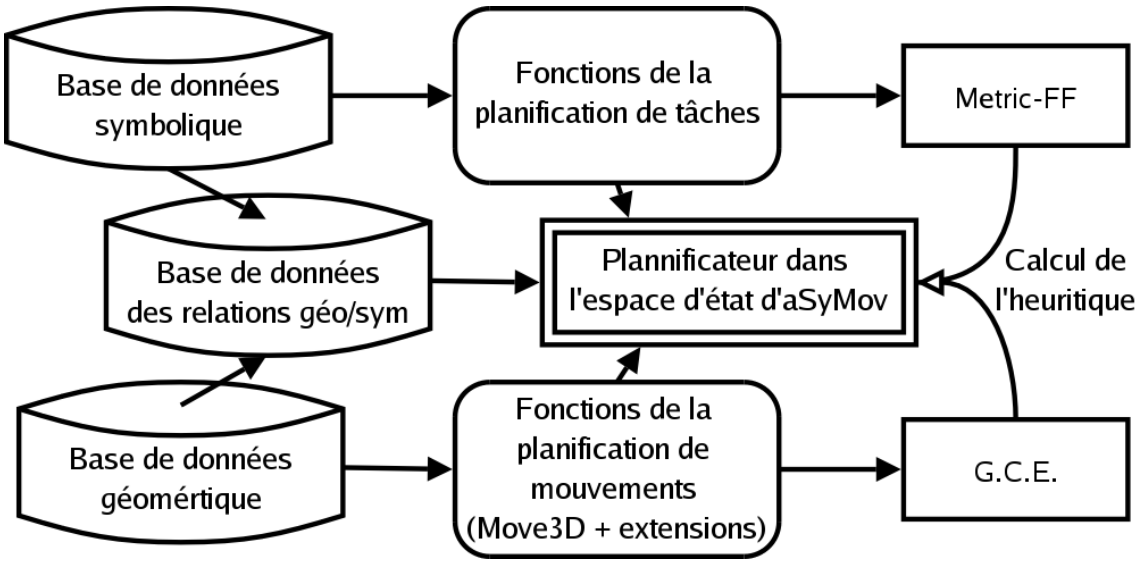


Figure 6.3: Architecture logicielle d'aSyMov.

- Une base de données pour la description géométrique du problème :
  - Description de l'environnement géométrique (définition 3D des obstacles et des robots et des objets déplaçables).
  - Description des cinématiques utilisées (méthodes locales, roadmaps) pour définir les mouvements possibles.
  - Description des liens entre roadmaps (§ 3.3) pour définir notamment les actions de prise et de pose.
  - Description des méthodes d'apprentissage de la topologie (roadmaps heuristiques (§ 4.1), règles heuristiques (§ 4.4)...).

- Une base de données pour la description symbolique du problème (§ 5.3) :
  - Description du domaine (types, actions,...) et des faits statiques.
  - Description de l'état initial et du but à atteindre.

- Une base de données de relations entre ces deux domaines :
  - Description des liens entre les noms utilisés dans les deux domaines sur des notions identiques (atome de type « `robot` » et nom de chaînes cinématiques, atome de type « `roadmap-type` » et nom de roadmap, ...) (ces relations sont décrites dans la section § 5.3.1).
  - Association de règles heuristiques aux instances des actions symboliques. Par exemple une action de déplacement d'un robot R dans une roadmap de *Transit* sera associée à une règle heuristique qui va étendre principalement la roadmap de *Transit* du robot R. Nous verrons comment ces associations sont utilisées pour guider l'apprentissage de la topologie (§ 6.4).

Remarque :
Une partie de ces données sont caractéristiques de la manipulation, elles peuvent donc être en partie construites automatiquement. En effet, si un robot peut prendre un objet, nous pouvons déjà en déduire l'utilisation de 4 roadmaps *Transit*, *Placement*, *Grasp* ∩ *Placement* et *Transfert* et les liens qui vont exister entre elles, nous pouvons aussi en déduire une partie de la définition du domaine symbolique avec les actions « `goto` », « `grasp` », « `ungrasp` » et « `switch` ».
Dans notre implémentation, seules les données symboliques profitent de cette génération automatique. La conception d'un nouveau domaine est encore assez fastidieuse, surtout pour les données géométriques.

6.2.1 L'algorithmique générale.

L'algorithmique générale d'aSyMov est assez simple, mais elle implique de nombreuses techniques de natures différentes. Les étapes principales sont : un prétraitement, une boucle de recherche dans l'espace d'état, et un post-traitement (figure 6.4.a).

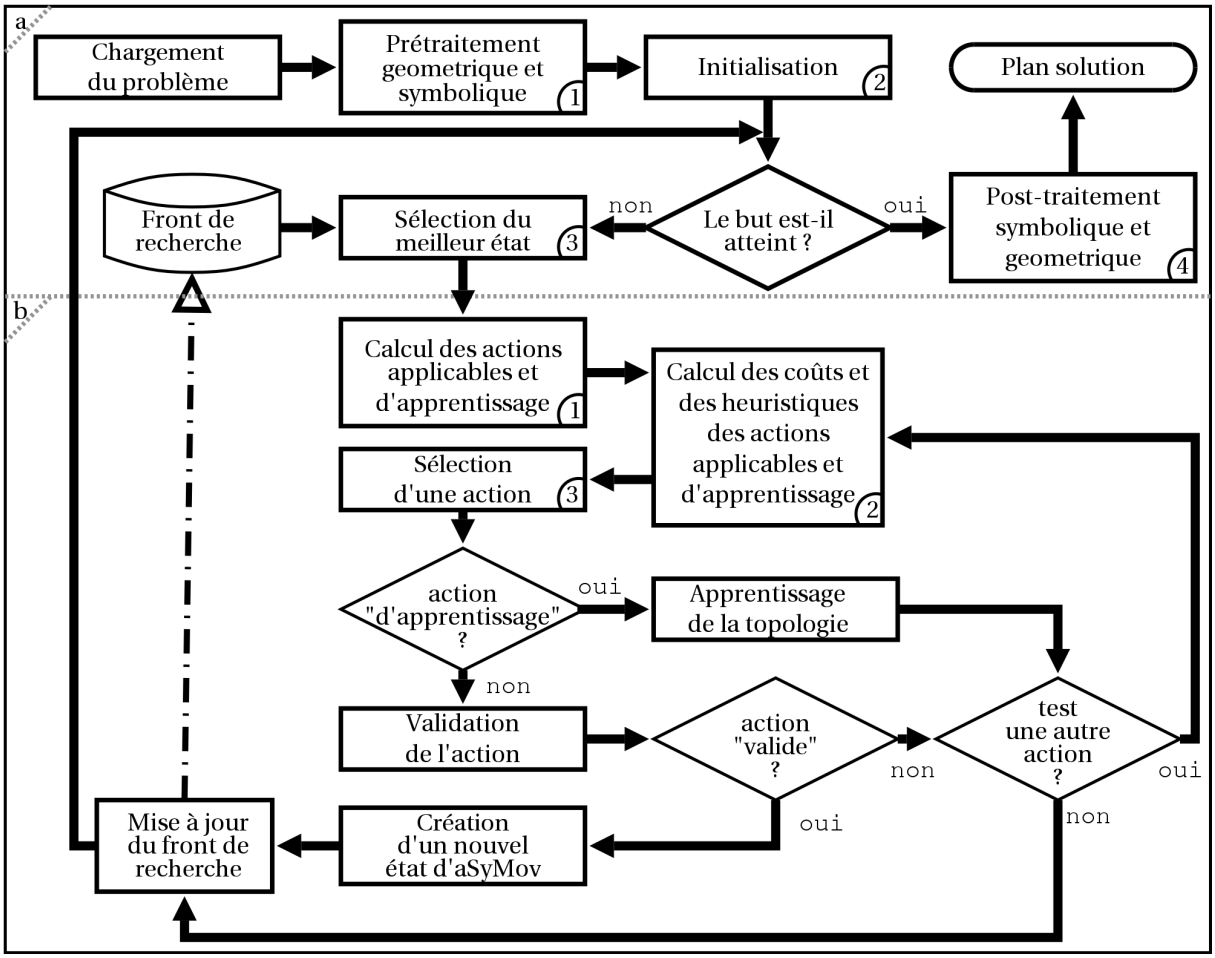


Figure 6.4: Algorithme général d'aSyMov. (a) représente les prétraitements, le choix des états à étendre, et le post-traitement. (b) représente les opérations effectuées sur un état sélectionné pour explorer de nouveaux états.

Le prétraitement (figure 6.4.a-1) est principalement fait par les fonctions de planification de tâches. Il va y avoir une analyse du domaine symbolique pour instancier les actions possibles et retirer les faits *inutiles*. Après cela aSyMov va en déduire les *« positions symboliques »* et les lier au domaine géométrique.

L'initialisation (figure 6.4.a-2) effectue une première phase d'enrichissement des roadmaps pour essayer de trouver pour chaque *position symbolique* du problème, au moins un noeud qui satisfasse ses critères.

Il peut y avoir aussi, une phase d'enrichissement supplémentaire pour essayer de trouver une connexité entre l'état initial et le but à travers les *« Graphes des Cinématiques Élémentaires »* (GCEs § 3.4). Toutefois, cette étape se faisant sans contexte particulier la topologie apprise n'est pas toujours utilisable dans les contextes des états parcourus par aSyMov (§ 6.4).

Une autre possibilité beaucoup plus efficace est de réutiliser les roadmaps apprises lors d'une recherche précédente.

Les opérations du post-traitement (figure 6.4.a-4) seront détaillées ultérieurement (6.6) et porteront sur : l'optimisation du plan tant au niveau symbolique qu'au niveau des trajectoires géométriques, la parallélisation des actions et la transformation des chemins des roadmaps affinables en des trajectoires valides.

La recherche d'une solution dans l'espace d'états peut être séparée en deux parties :

1. La sélection d'un état à partir duquel seront appliquées des actions (figure 6.4.a-3) (§ 6.2.2).
2. L'extension de l'espace de recherche appliquée à l'état sélectionné (figure 6.4.b) et qui concerne à la fois l'apprentissage de la topologie (§ 6.4) et la recherche de nouveaux états valides (§ 6.2.3).

6.2.2 Le choix des états à étendre.

Nous avons choisi l'algorithme A* [Nilson 80] pour sélectionner l'état à étendre (figure 6.4.a-3), c'est à dire le choix de l'état ayant le coût total le plus faible. Il y a toutefois une différence par rapport à l'algorithme classique. ASyMov ne connaît pas parfaitement la topologie de l'environnement.

Les techniques de planification de mouvements probabiliste ne peuvent fournir que deux réponses : « j'ai trouvé un chemin » ou « je ne sais pas s'il existe un chemin ». Dans le premier cas, il n'y a pas de problème. Par contre, dans le second cas, l'action qui a échoué pourra être reconsidérée ultérieurement quand les roadmaps auront été enrichies.

C'est pourquoi les actions et les états dans aSyMov ont trois coûts au lieu de deux :

- Coût propre : coût pour effectuer une action ou pour atteindre un état.
- Coût heuristique : estimation du coût du chemin qu'il reste à faire après avoir fait cette action ou atteint cet état.
- Coût des échecs : c'est le coût supplémentaire que nous avons rajouté aux états et aux actions d'aSyMov. Au lieu de supprimer une action qui n'est pas applicable avec la connaissance actuelle

de la topologie, nous avons décidé d'augmenter son coût. Cela va permettre de favoriser d'autres actions ou d'autres états tout en laissant la possibilité de retenter cette action.

Ainsi le *« coût total »* qui servira à comparer les états entre eux (et plus tard les actions entre elles) est égal à la somme des trois coûts précédents.

Une seconde difficulté est que même si deux *états d'aSyMov* $\mathcal{E}_1$ et $\mathcal{E}_2$ ont un même *état symbolique* $\mathcal{E}_S$, et même si leurs *état d'accessibilité* sont identiques, il se peut que $\mathcal{E}_1$ et $\mathcal{E}_2$ soient différents, pourvu qu'il existe des *états géométriques* non communs (figure 6.2).

Ainsi, par rapport à un espace d'état classique, l'espace d'état d'aSyMov est donc beaucoup plus grand et les actions infaisables à un moment donné doivent pouvoir être considérées à nouveau.

Il est délicat de savoir si deux *états d'aSyMov* ayant un même *état symbolique* sont fondamentalement différents, ou si aSyMov a fait un cycle dans l'espace d'état. Ceci n'est en fait possible qu'avec une connaissance exacte de la topologie du problème. Il se trouve qu'expérimentalement des cycles inutiles, dans l'espace d'état, apparaissent parfois. De futurs travaux vont tenter d'y apporter une solution satisfaisante.

Dans le cadre de cette thèse nous avons choisi une méthode plus empirique qui consiste à défavoriser la sélection des *états d'aSyMov* quand il existe plusieurs autres états ayant le même *état symbolique*. Pour cela, nous incluons dans le coût des échecs des états un coût proportionnel au nombre *d'états d'aSyMov* ayant le même *état symbolique*.

Nous avons observé expérimentalement que cette méthode empirique donnait d'assez bons résultats et diminuait le nombre d'états explorés inutilement.

Pour résumer :

Pour une action applicable mais pas encore validée (au sens géométrique) :

- Coût propre : valeur propre à l'action. Dans l'implémentation actuelle d'aSyMov cette valeur vaut 1.
- Coût heuristique : estimation du coût des actions restantes. Dans l'implémentation actuelle d'aSyMov cette valeur est la longueur du plan symbolique restant, corrigée par la géométrie (§ 6.5)
- Coût des échecs : coût incrémental en fonction des échecs de validation de l'action dans le domaine géométrique. Dans l'implémentation actuelle d'aSyMov cette valeur est une fonction linéaire du nombre d'échecs.

Pour un état d'aSyMov dont toutes les actions n'ont pas encore été validées :

- Coût : coût du plan nécessaire pour atteindre cet état depuis l'état initial.
- Coût heuristique : Coût total minimal de l'ensemble des actions applicables, mais non encore validées sur cet état.
- Coût des échecs : coût incrémental en fonction du nombre d'état d'aSyMov avec un état symbolique identique plus un coût proportionnel au nombre d'échecs de l'algorithme d'extension de l'état.

Remarque :
Le coût heuristique des états dépend du coût heuristique des actions qui lui-même dépend de la connaissance actuelle de la topologie. Comme l'apprentissage de cette topologie se fait tout au long de la planification, ce coût peut ne plus être valide. Mais une fonction qui recalculerait l'ensemble de ces coûts serait gourmande en temps de calcul alors qu'ils ne représentent que des estimations des coûts réels.

6.2.3 L'extension de l'espace d'état.

L'algorithme 1 représente les étapes de l'extension de l'état (figure 6.4.b).

```
1:  nouvel-état ← ∅
2:  Calcul_Actions_Applicables()
3:  Calcul_Actions_Apprentissage()
4:  Répéter
5:    Calcul_Coûts_Heuristiques_Actions()
6:    action ← Choix_Action()
7:    Si (Est_Action_Apprentissage(action)) Alors
8:      Enrichissement_Roadmap(action)
9:    Sinon
10:     Si (Est_Valide(action)) Alors
11:       nouvel-état ← Créer_État(action)
12:     Fin Si
13:   Fin Si
14:   Mise_A_Jour_Coût_Échec(action)
15: Jusqu'à (Fin_Essais())
```

Algorithme 1: Algorithme d'extension d'un état sélectionné.

La fonction « `Calcul_Actions_Applicables` » (algorithme 1 ou figure 6.4.b-1) détermine les actions applicables à partir de *l'état symbolique* de l'état sélectionné. Une action est dite applicable si elle vérifie toutes les préconditions symboliques. Nous utilisons pour ce faire des fonctions du planificateur symbolique (*« Metric-FF »*).

Cette action peut avoir des conséquences géométriques, c'est-à-dire qu'il y a ajout ou retrait de prédicats « `on` ». Dans ce cas il y a changement de position symbolique et donc soit un déplacement, soit l'utilisation de liens entre roadmaps. Nous dirons que c'est une *« action géométrique »*.

Une *« action géométrique »* va impliquer l'utilisation des roadmaps. Dans ce cas il se peut qu'elle soit applicable mais non valide car il ne sera pas possible de trouver un chemin à travers les roadmaps. Si on ne trouve pas un chemin à travers les roadmaps cela peut vouloir dire deux choses : ou bien cette action est infaisable ou bien il faut enrichir la topologie capturée par les roadmaps pour pouvoir trouver une solution.

Afin d'enrichir la topologie des roadmaps nous allons concevoir des actions d'enrichissement de roadmap. L'algorithme choisira une action qui pourra être soit enrichir ses connaissances sur la géométrie du problème, soit chercher une solution avec son niveau actuel de connaissance. Nous appellerons de telles actions des *« actions d'apprentissage »*.

La fonction « `Calcul_Actions_Apprentissage` » permet justement d'élaborer ces actions. Il existe plusieurs types *« d'actions d'apprentissage »* (§ 6.4.2) mais toutes font référence à des *« actions géométriques »* puisque le but est d'enrichir les roadmaps pour appliquer ces actions. Une action couramment utilisée est une action dite de *« recherche en profondeur »* qui va vouloir enrichir les roadmaps des actions géométriques applicables d'un plan solution heuristique.

Une fois que l'on a cet ensemble d'actions, il faut pouvoir les comparer, et pour cela, aSyMov calcule les coûts totaux de toutes les actions (`Calcul_Coûts_Heuristiques_Actions` et figure 6.4.b-2) . Nous verrons plus en détail ce point un peu plus loin (§ 6.5).

Après cela nous sélectionnons de manière pseudo-aléatoire une action à appliquer (figure 6.4.b-3 et `Choix_Action`). Nous allons dans un premier temps pondérer ces actions en fonction de leur coût total pour que l'action ayant le coût le plus faible soit l'action ayant le plus de chance d'être sélectionnée.

Dans la plupart des exemples que nous avons développés, nous essayons de montrer qu'aSyMov est capable de résoudre des problèmes où le géométrique a une forte influence sur le symbolique, autrement dit les cas où les heuristiques qu'il calcule ne sont pas parfaites. C'est pour cette raison que nous avons introduit une partie aléatoire à ce niveau afin d'améliorer les performances de l'algorithme.

Cette partie aléatoire représente en fait notre incertitude quant à la justesse de nos évaluations des coûts heuristiques.

Une fois l'action sélectionnée, nous allons tenter de l'appliquer. Si c'est une *« action d'apprentissage »*, nous allons ajouter de nouveaux nœuds aux roadmaps (« `Enrichissement_Roadmap` » ou figure 6.4.b-4) (§ 6.4). Sinon nous allons essayer de la valider au niveau géométrique (« `Est_Valide` » ou figure 6.4.b-5) (§ 6.3).

Nous allons répéter ces opérations jusqu'à ce que nous trouvions une action applicable valide ou jusqu'à satisfaire un critère de fin (généralement un temps maximal ou un nombre de cycles maximal).

6.3 Le processus de validation des positions.

Afin de limiter la complexité des roadmaps construites, elles sont générées sans prendre en compte toutes les positions possibles des autres robots et objets de l'environnement. C'est le rôle du processus de validation de vérifier si les chemins possibles dans les roadmaps peuvent s'appliquer en tenant compte des autres robots (ou objets).

Ce processus va rechercher à atteindre de nouveaux *« états géométriques »* $\mathcal{E}_{G_i} = (N_1, \ldots, N_n)$ *inclus*[2] dans l'état d'accessibilité $\mathcal{E}_A = (Acc_1, \ldots, Acc_n)$.

[2] $N_1 \in Acc_1, \ldots, N_n \in Acc_n$

1: $\mathcal{E}_{i-1} \leftarrow$ `Précédent`$(\mathcal{E}_i)$
2: $\mathcal{C}_{i-1} \leftarrow$ `Précédent`$(\mathcal{C}_i)$
3: **Tant Que** (*VRAI*) **Faire**
4: **Pour Tout** $\mathcal{E}_{i-1\,G_j} \in \mathcal{E}_{i-1}$ / $\mathcal{E}_{i-1\,G_j} \notin \mathcal{C}_{i-1}$ **Faire**
5: **Si** (`Valide`$(\mathcal{A}_i, \mathcal{E}_{i-1\,G_j}, \mathcal{C}_i)$) **Alors**
6: `Ajout_Nouvel_Etat_Geo`$(\mathcal{E}_i, \mathcal{A}_i, \mathcal{E}_{i-1\,G_j}, \mathcal{C}_i)$
7: **Retourne** *VRAI*
8: **Sinon**
9: `Ajout_Contraintes`$(\mathcal{C}_{i-1}, \mathcal{A}_i, \mathcal{E}_{i-1\,G_j})$
10: **Fin Si**
11: **Fin Pour**
12: **Si** $((\mathcal{E}_{i-1\,A} \subset \mathcal{C}_{i-1})$ ou ($\mathcal{E}_{i-1}$ est l'état initial)) **Alors**
13: **Retourne** *FAUX*
14: **Fin Si**
15: $\mathcal{A}_{i-1} \leftarrow$ `Précédent`$(\mathcal{A}_i)$
16: **Si** ($\neg$ `Validation`$(\mathcal{E}_{i-1}, \mathcal{A}_{i-1}, \mathcal{C}_{i-1})$) **Alors**
17: **Retourne** *FAUX*
18: **Fin Si**
19: **Fin Tant Que**

Algorithme 2: `Validation`$(\mathcal{E}_i, \mathcal{A}_i, \mathcal{C}_i)$

Autrement dit, quand aSyMov voudra valider une *« action géométrique »*, il essaiera d'atteindre une combinaison des nœuds des roadmaps correspondantes aux positions symboliques appliquées du nouvel état[3] pour créer un nouvel état géométrique $\mathcal{E}_{G_i}$. Il gardera toutefois l'ensemble des nœuds possibles dans les listes d'accessibilités $\mathcal{E}_A$.

Avec ce mécanisme nous pourrons aussi remettre en cause des choix géométriques précédemment faits s'ils entrent en conflit avec la création du nouvel état géométrique. Par exemple, si une action précédente a déplacé un objet et si cet objet obstrue le mouvement qu'aSyMov essaye de valider, aSyMov peut changer le choix qu'il a fait et poser l'objet sur un autre nœud de la liste d'accessibilités de la position symbolique appliquée de pose de l'objet.

6.3.1 Algorithme de la validation.

L'algorithme 2 montre le processus de validation des actions. C'est une fonction récursive, qui prend comme paramètres $\mathcal{E}_i$ *« l'état d'aSyMov »* créé par l'action $\mathcal{A}_i$ et les contraintes $\mathcal{C}_i$. Les contraintes $\mathcal{C}_i$ représentent l'ensemble des *« états géométriques »* qui ne doivent pas être pris en compte.

Pour la validation d'une nouvelle action $\mathcal{A}_i$, $\mathcal{C}_i = \emptyset$ et *« l'état d'aSyMov »* $\mathcal{E}_i$ est créé, mais il ne

[3] Un *« état d'aSyMov »* est $\mathcal{E} = (\mathcal{E}_S, \mathcal{E}_A, \{\mathcal{E}_{G_1}, \ldots, \mathcal{E}_{G_n}\})$ (§ 6.1)

possède aucun état géométrique. La fonction de `Validation` lui créera un état géométrique valide en cas de succès.

Tant que c'est possible (`Valide`) la fonction `Ajout_Nouvel_Etat_Geo` crée un nouvel état géométrique et la fonction de `Validation` est alors un succès.

Sinon des contraintes sur la sélection des états géométriques sont rajoutées par `Ajout_Contraintes`. Cette fonction est très importante. Elle doit au minimum permettre de rejeter les états géométriques déjà testés. Donc si $\mathcal{E}_{i-1\,G_j}$ est invalide la fonction doit au moins assurer que $\mathcal{E}_{i-1\,G_j} \in \mathcal{C}_{i-1}$.

Dans certains cas, une action de déplacement d'un robot `R` échoue à cause d'un seul objet `O`. A ce moment, il est possible de dire que les noeuds des positions du robot `R` et de l'objet `O` sont incompatibles. Il est alors possible d'augmenter les contraintes en interdisant la combinaison de ces deux noeuds.

Dans les autres cas `Ajout_Contraintes` n'ajoute à $\mathcal{C}_{i-1}$ que les états géométriques déjà testés.

Si les contraintes sont si fortes qu'elles incluent l'état d'accessibilité, alors il est impossible de trouver une solution. Sinon il est possible de propager les contraintes sur l'état précédent afin de trouver de nouveaux états géométriques.

Remarque :
La fonction la plus coûteuse de cet algorithme est celle qui permet de tester l'existence d'un chemin dans un contexte donné « `Valide` ». C'est pourquoi elle a été particulièrement optimisé.

La recherche de ce chemin se fait en deux phases. Une première phase utilise un algorithme de type parcours en largeur de la roadmap pour noter les noeuds en fonction de la distance qu'ils ont par rapport aux buts[4]. Cette distance étant multipliée par la fragilité des arcs[5].

La seconde phase va vérifier la validité des arcs de la roadmap dans le contexte pris en question. Le chemin à valider étant celui qui minimise notre *« distance »* aux buts. En cas d'invalidité d'un arc, ce défaut sera propagé pour mettre à jour les *« distances »*.

Cet algorithme réduit grandement le nombre de chemins locaux à tester par rapport à l'algorithme A* [Nilson 80]. Mais pour réduire encore ce facteur, nous utilisons un marquage des arcs des roadmaps. Chaque arc contiendra une liste des noeuds des autres robots ou objets avec lesquels il n'entre pas en collision et une liste de ceux avec qui il est invalide[6]. Ce marquage permet de réutiliser les tests de collision déjà effectués.

La combinaison de ces deux techniques nous a fait gagner un facteur 100 pour la vitesse de la fonction « `Valide` ».

[4]Nœuds qui satisfont les contraintes de la position symbolique finale de l'action à valider.

[5]Les arcs souvent invalides ou peu testés seront défavorisés par rapport aux autres

[6]ASyMov ne bouge qu'un seul robot (éventuellement composé par d'autres robots) à la fois, les autres sont placés aux positions de l'état géométrique précédent, c'est à dire sur des nœuds de roadmaps.

6.3.2 Exemple de validation des actions.

Nous allons illustrer l'algorithme de validation à travers un exemple simple de deux chariots élévateurs et souligner comment il peut remettre en cause des choix géométriques précédemment faits (§ 6.3.3). La figure 6.6 montre les listes d'accessibilités qui sont progressivement construites pendant la recherche du plan illustré par la figure 6.5.

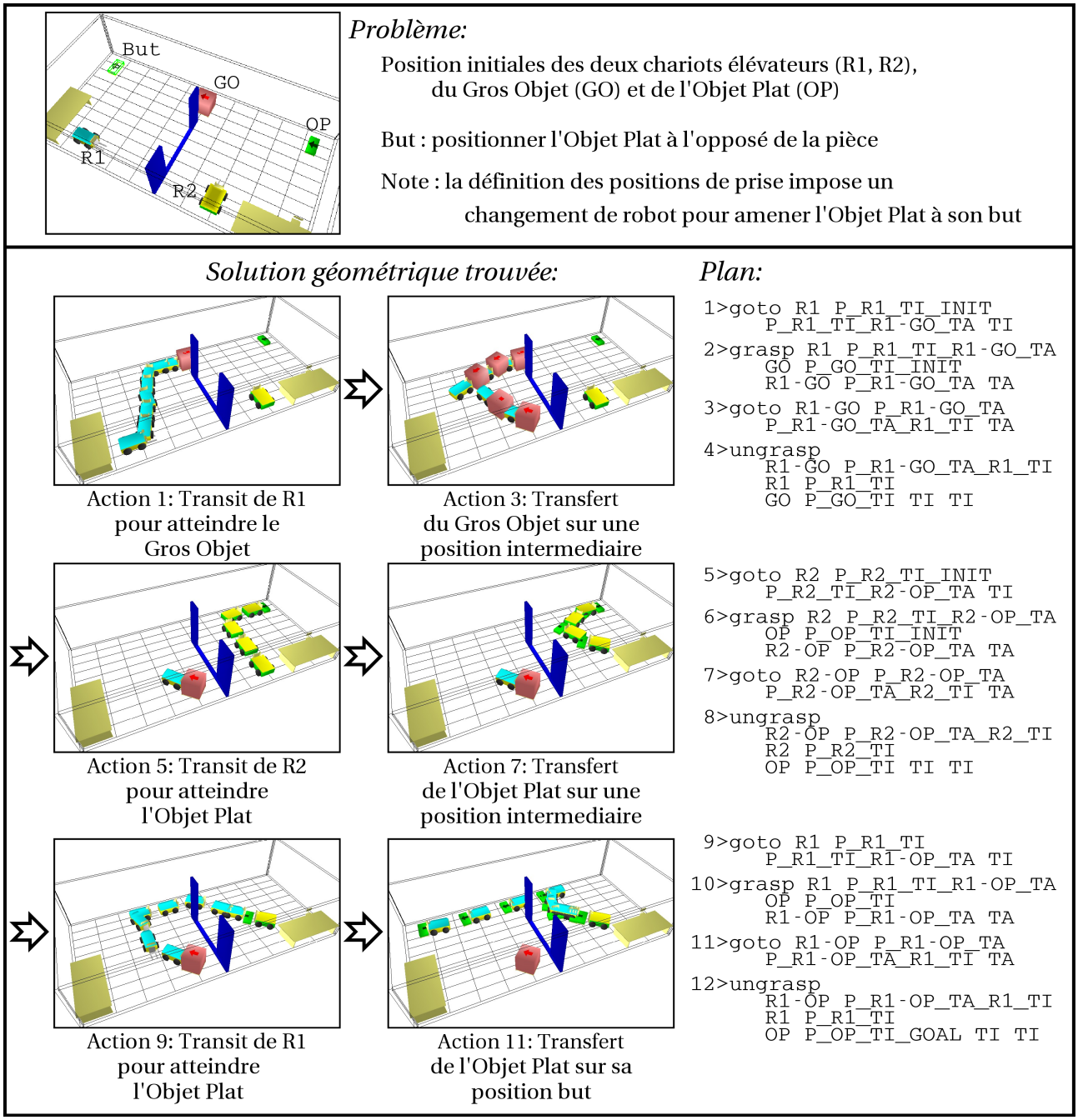


Figure 6.5: Exemple de deux robots chariots élévateurs (« `R1` » et « `R2` ») qui doivent porter un objet plat « `OP` » dans l'autre salle. La figure du haut montre l'environnement et les positions initiales et finales. La deuxième partie montre le plan solution qui a été produit par aSyMov (12 étapes). Les trajectoires des robots sont illustrées dans la partie gauche.

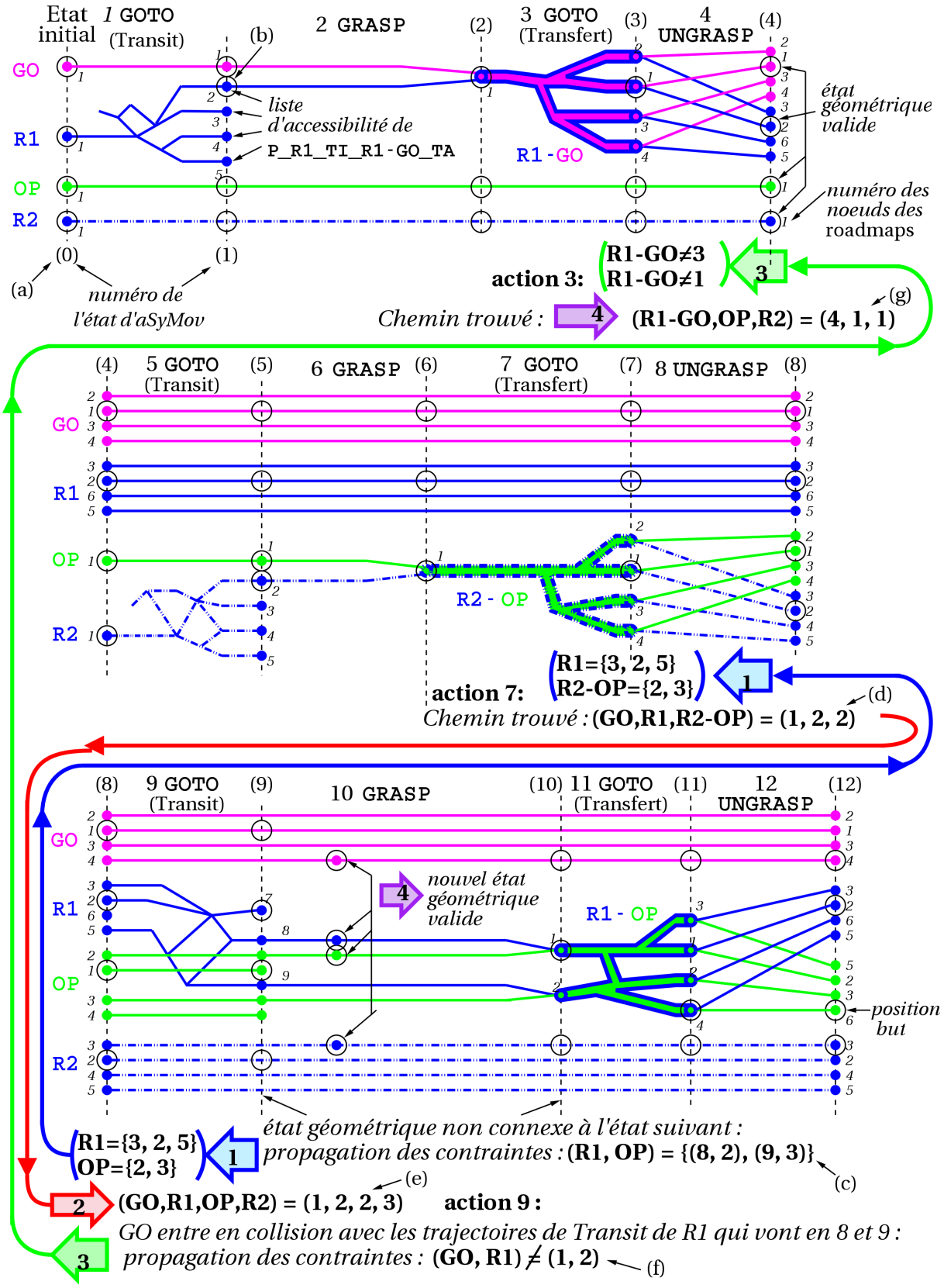


Figure 6.6: Illustration de la gestion des listes d'accessibilité et de l'instanciation des positions. La figure montre toutes les actions du plan de la figure 6.5. La validation de l'action 10 implique une propagation arrière de l'accessibilité pour valider de nouveaux états géométriques dans les états précédents.

Dans l'état initial $\mathcal{E}_0$, une seule configuration (nœud de roadmap) est donnée pour chaque robot/objet. $\mathcal{E}_0$ n'a qu'un seul état géométrique $\mathcal{E}_{0\,G_1} = (1, 1, 1, 1)$ correspondant aux noeuds 1 de `GO`, `R1`, `OP` et `R2` (figure 6.6.a).

L'action 1 est une action de *Transit* pour le robot `R1`. Elle permet d'atteindre les nœuds appartenant à la roadmap de *Transit* de `R1` qui sont dans la composante connexe de la configuration initiale du robot et qui liés à des nœuds de la roadmap de *Transfert* du robot `R1` saisissant le gros objet (`R1-GO`). Ces nœuds formeront la liste d'accessibilité de la position symbolique appliquée suivante.

Afin que l'action 1 soit applicable, le planificateur doit trouver au moins un chemin valide (sans collision) pour `R1`. Supposons qu'aSyMov ait choisi un chemin qui le mène au noeud 2 (figure 6.6.b). L'état géométrique $\mathcal{E}_{1\,G_1} = (1, 2, 1, 1)$ correspondant à l'état symbolique $\mathcal{E}_{1\,S}$ avec les faits `(on GO P_GO_TI_INIT)`, `(on R1 P_R1_TI_R1-GO_TA)`, `(on OP P_OP_TI_INIT)` et `(on R2 P_R2_TI_INIT)`.

L'action 2 est la prise de « `GO` » par `R1`. Cela correspond un changement de roadmaps pour avoir le mouvement du robot composé « `R1-GO` ». Dans notre exemple, seul le nœud 2 de `R1` permet de saisir `GO` dans sa position initiale. Les autres noeuds sont connectés à d'autres positions de l'objet.

Si l'état géométrique ne contenait pas le noeud 2 de `R1` il faudrait effectuer une propagation arrière (§ 6.3.3) pour trouver un nouvel état géométrique contenant le noeud 2.

La liste d'accessibilité de la position symbolique appliquée suivante n'a qu'un seul élément. Puisque les nœuds précédents des roadmaps de `R1` et `GO` sont dans un état géométrique valide, le nœud composé est également valide. Aucun test de collision ne sera fait pour valider le nouvel état géométrique. L'état géométrique $\mathcal{E}_{2\,G_1} = (1, 1, 1)$ correspondant à l'état symbolique $\mathcal{E}_{1\,S}$ avec les faits `(on R1-GO P_R1-GO_TA)`, `(on OP P_OP_TI_INIT)` et `(on R2 P_R2_TI_INIT)`.

La troisième action est semblable à la première (« `goto` »), mais elle implique le robot composé `R1-GO` qui doit arriver à une position où il peut libérer `GO` (prise au sol). Comme le montre la figure 6.6, la roadmap de *Transfert* autorise plusieurs positions finales possibles.

A cet instant de la recherche, il n'y a pas aucune autre contrainte sur le choix de la position finale à valider, un choix « paresseux » est donc fait. ASyMov cherche à valider un nouvel état géométrique qui demande le moins de tests de collisions. Dans ce cas, il ne va même pas déplacer `GO` puisque la position de saisie permet aussi de le lâcher. L'état géométrique est le même : $\mathcal{E}_{3\,G_1} = \mathcal{E}_{2\,G_1} = (1, 1, 1)$, mais il correspond à un état symbolique différent : `(on R1-GO P_R1-GO_TA_R1_TI)`, `(on OP P_OP_TI_INIT)` et `(on R2 P_R2_TI_INIT)`

La quatrième action permet de lâcher `GO`. C'est une action de décomposition du robot `R1-GO` en robot `R1` et objet `GO` correspondant à un changement de roadmap. Elle ne nécessite aucun test géométrique particulier. $\mathcal{E}_{4\,G_1} = (1, 2, 1, 1) = \mathcal{E}_{2\,G_1}$

Les prochaines actions 5 à 8 sont similaires aux 4 premières (figure 6.6). Le robot `R2` se déplace pour transporter l'objet plat « `OP` ». Les tests de validité des actions sont faits en prenant en compte les positions du robot `R1` et de l'objet `GO` trouvées précédemment. Finalement l'état $\mathcal{E}_8$ possède un état géométrique $\mathcal{E}_{8\,G_1} = (1, 2, 1, 2)$.

L'action 9 est elle aussi similaire à la première action. Nous avons donc $\mathcal{E}_{9\,G_1} = (1, 7, 1, 2)$.

Cependant la prochaine étape ne sera pas faisable à partir de $\mathcal{E}_{9\,G_1}$. Nous allons voir comment nous allons remettre en cause les choix précédents pour créer de nouveaux états géométriques permettant d'appliquer l'action 10.

6.3.3 La propagation arrière.

Quand il n'est pas possible de valider une action avec un des états géométriques de l'état courant, aSyMov va essayer de construire un nouvel état géométrique pour permettre l'application de cette action. Pour cela, il va en quelque sorte propager les contraintes de connexité et d'anti-collision sur les états précédents.

Les changements ne porteront que sur les états géométriques. Ils ne remettront pas en cause ni la partie symbolique des états, ni les listes d'accessibilités des positions symboliques appliquées des états.

La 10^{e} action permet la saisie de `OP` par `R1` et correspond à un changement de roadmaps.

Il se trouve qu'il existe des nœuds dans les listes d'accessibilité des positions symboliques appliquées du nouvel état résultant. Par contre ces nœuds ne sont pas connexes avec les nœuds de l'état géométrique précédent $\mathcal{E}_{9\,G_1}$. ASyMov va donc essayer de créer des états géométriques pour les états précédents qui vont être connexes avec les listes d'accessibilité du nouvel état.

Pour cela il va propager les contraintes de composition des noeuds (figure 6.6.c). Il est possible de créer un nouvel état géométrique pour l'action 10 s'il existe un $\mathcal{E}_{9\,G_i}$ du type $(x, 8, 2, y)$ ou $(x, 9, 3, y)$.

Cette contrainte va être propagée à l'état 8 à travers l'action 9. Il y a 3 noeuds qui mènent aux noeuds 8 et 9. Donc il faut trouver un $\mathcal{E}_{8\,G_i} = (w, x, y, z)$ avec $x \in \{3, 2, 5\}$ et $y \in \{2, 3\}$.

Cette contrainte va être propagée jusqu'à l'état 7, où nous aurons $\mathcal{E}_{7\,G_i} = (w, x, y)$ avec $x \in \{3, 2, 5\}$ et $y \in \{2, 3\}$. Or l'action 7 permet justement d'atteindre un état géométrique qui satisfait ces contraintes. Il y a deux possibilités $(1, 2, 2)$ et $(1, 2, 3)$.

Supposons que le nouvel état géométrique trouvé soit $\mathcal{E}_{7\,G_2} = (1, 2, 2)$ (figure 6.6.d). Il y a alors propagation en avant des nouveaux états géométriques créés : $\mathcal{E}_{8\,G_2} = (1, 2, 2, 3)$ (figure 6.6.e).

Malheureusement `OP` et `R1` se trouvent encore dans deux pièces différentes avec `GO` qui bloque le passage. Ainsi l'action 9 ne permet pas de trouver un chemin vers les noeuds 7 et 8 en partant du noeud 2 et avec `GO` sur le noeud 1. Il faut de nouveau trouver un autre état géométrique précédent auquel aSyMov rajoute la contrainte de ne pas avoir simultanément `R1` en 2 et `GO` en 1. $\mathcal{E}_{8\,G_i} = (w, x, y, z)$ avec $x \in \{3, 2, 5\}$, $y \in \{2, 3\}$ et $(w, x) \neq (1, 2)$.

Cette contrainte va de nouveau être propagée (figure 6.6.f) jusqu'à l'état 3. Les contraintes $x \in \{3, 2, 5\}$ et $(w, x) \neq (1, 2)$ de $\mathcal{E}_{8\,G_i}$ deviendront pour $\mathcal{E}_{3\,G_i} = (x, y, z)$, $x \neq 3$ et $x \neq 1$.

Supposons que le nouvel état géométrique trouvé soit $\mathcal{E}_{3\,G_2} = (4, 1, 1)$ (figure 6.6.g). Il y a de nouveau propagation en avant des états géométriques créés : $\mathcal{E}_{4\,G_2} = (4, 5, 1, 1)$, $\mathcal{E}_{5\,G_2} = (4, 5, 1, 2)$, $\mathcal{E}_{6\,G_2} = (4, 5, 1)$, $\mathcal{E}_{7\,G_3} = (4, 5, 2)$, $\mathcal{E}_{8\,G_3} = (4, 5, 2, 3)$.

Cette fois-ci `GO` n'est plus gênant, il est possible de trouver un chemin faisable pour l'action 9 pour créer le nouvel état géométrique $\mathcal{E}_{9\,G_2} = (4, 8, 2, 3)$.

Ce nouvel état permet d'appliquer l'action 10 puis les actions 11 et 12 pour finalement atteindre la solution.

6.3.4 Conclusion sur la validation.

Le procédé de validation est une opération coûteuse, qui peut, dans le pire cas, être exponentielle avec le nombre de robots. L'implémentation actuelle est complète, elle assure de trouver une solution si elle existe au sein des roadmaps. De ce fait sa complexité croit aussi linéairement en fonction de chaque liste d'accessibilités.

Par contre comme l'a montré l'exemple précédent, quand il existe des solutions le processus peut être très économe en tests de collisions. Les objets ne sont déplacés que si nécessaire et les contraintes réduisent progressivement l'espace de recherche de ces solutions.

Cet algorithme est la clef de voûte de la planification géométrique dans l'espace d'état. En effet, l'espace d'état géométrique correspond au produit cartésien des noeuds des roadmaps. Il est beaucoup trop important pour faire une simple recherche en avant.

Cet algorithme combine en fait trois techniques pour restreindre la recherche : la construction d'un état d'accessibilité (en avant), une propagation de contrainte (en arrière), une recherche dans l'espace des états géométriques (en avant).

6.4 L'enrichissement des roadmaps.

La résolution d'un problème complet impliquant plusieurs robots, exige l'emploi d'un grand nombre de roadmaps spécialisées. Il y a naturellement les roadmaps de mouvement (*Transit*, *Transfert*,...) mais aussi des roadmaps heuristiques qui aident à l'étude de la topologie du problème pour améliorer la construction des roadmaps de mouvement (§ 3.2.1). Comme nous l'avons dit précédemment nous allons étudier ici comment va se faire le choix des roadmaps à enrichir et des méthodes utilisées. Mais dans un premier temps nous allons faire quelques remarques sur les choix que nous avons faits pour aSyMov.

6.4.1 L'enrichissement des roadmaps au cours de la planification.

ASyMov a été conçu pour enrichir les roadmaps pendant qu'il fait la recherche d'un plan valide dans l'espace d'état. Il aurait pu être envisagé de faire une première étape uniquement géométrique sur la construction de ces roadmaps, suivie d'une étape de recherche dans l'espace d'état. En fait, aSyMov peut très bien charger des roadmaps déjà apprises et les réutiliser pour faire une nouvelle recherche dans l'espace d'état sans les enrichir. Toutefois nous n'avons pas développé des méthodes qui permettraient à aSyMov de construire entièrement ces roadmaps avant la recherche dans l'espace d'état et ceci principalement pour deux raisons :

- L'ajout de nœuds sans contexte particulier : si aSyMov enrichit les roadmaps avant sa recherche dans l'espace d'état, il ne peut le faire que s'il ne prend pas en compte les positions des autres robots et objets. En effet, les contextes dans lequel doit être validé une roadmap seront définis

par les états géométriques trouvés par aSyMov au cours de la planification. Ils ne seront pas réellement connus a priori. Or il peut être intéressant de savoir qu'une porte ouverte sera fermée plus tard dans le plan et qu'il faudra peut être trouver d'autres passages dans la roadmap.

- Le choix des roadmaps à enrichir : si aSyMov enrichit les roadmaps pendant qu'il effectue sa recherche dans l'espace état, il peut connaître les actions qui vont lui être réellement utiles et par conséquent les roadmaps qui doivent être enrichies. Cela peut représenter un sous-ensemble des roadmaps du problème et limiter ainsi beaucoup la recherche.

Un autre choix possible pour aSyMov, serait de lancer une opération complète de planification de mouvement pour chaque action. Mais cette option aurait aussi deux inconvénients :

- Les critères de fin : il pourrait être délicat de décider quand on a un échec, ou quand il faut enrichir davantage la roadmap.
- La réutilisation : il peut-être intéressant de réutiliser une roadmap déjà apprise dans un contexte donné comme base pour l'élaboration de la roadmap dans un nouveau contexte. En effet les arcs des roadmaps seront au moins valides par rapport à l'environnement statique.
  Pour pouvoir réutiliser efficacement les roadmaps dans différents contextes, chaque arc d'une roadmap gardera en mémoire un certain nombre de configurations des autres robots ou objets avec lesquelles il a été validé ou invalidé.

C'est pour toutes ces raisons, qu'aSyMov réutilise ses roadmaps et essaie de les revalider ou de les enrichir dans un contexte donné. Un cas d'échec du planificateur de mouvement n'implique en rien la non faisabilité du mouvement, cette roadmap sera prospectée ultérieurement si un autre chemin plus intéressant n'a pas été trouvé entre-temps. Généralement aSyMov ajoute un faible nombre de nœuds à chaque fois, ce qui permet d'explorer plusieurs états différents, donc plusieurs roadmaps.

S'il existe plusieurs successions d'actions qui mènent au but et que ces actions se font sur différentes roadmaps, aSyMov peut explorer simultanément ces actions dans le cas où l'apprentissage de roadmaps serait délicat. En effet dans ce cas les échecs risquent d'être nombreux et ainsi le coût des échecs va combler la différence qu'il peut y avoir au niveau des coûts heuristiques.

6.4.2 Les actions d'apprentissage.

Pour enrichir les roadmaps pendant la recherche d'un plan dans l'espace d'état, nous avons fait le choix d'ajouter aux actions applicables des actions dites *« d'apprentissage »*.

L'introduction de cet apprentissage de la topologie par l'utilisation d'actions est un moyen simple d'avoir plusieurs stratégies pour modifier les roadmaps qui vont dépendre des actions que l'on veut accomplir. De plus ce choix permet de mettre en lumière l'équilibre qui doit se faire entre l'enrichissement de la topologie ou le choix de la recherche d'un plan à partir des connaissances déjà acquises.

Comme toutes les actions, ces actions ont aussi trois coûts :

- Coût propre : valeur relative à la difficulté d'ajouter de nouveaux nœuds. Dans l'implémentation actuelle d'aSyMov cette valeur est fixée à 2. A priori il serait plus avantageux d'avoir un

coût dépendant du temps moyen passé pour ajouter un nœud, mais il est difficile d'avoir une conversion efficace entre un temps passé et un nombre d'actions (unité du coût propre).

- Coût heuristique : estimation de l'intérêt de l'action. Dans l'implémentation actuelle d'aSyMov cette valeur est la longueur du plan symbolique restant, corrigée par la géométrie (§ 6.5).
- Coût des échecs : coût incrémental en fonction du nombre d'applications de cette action d'apprentissage. Dans l'implémentation actuelle d'aSyMov cette valeur est une fonction linéaire du nombre d'utilisations de cette action.

Actuellement seulement deux types *« d'actions d'apprentissage »* sont implémentées dans aSyMov :

- La *« recherche en profondeur »* : cette action permet d'enrichir les roadmaps pour faciliter l'application des actions d'un plan heuristique trouvé par le planificateur symbolique.
- La *« recherche en largeur »* : cette action permet d'enrichir toutes les roadmaps utilisées par les actions applicables à l'état courant.

Ces actions permettent de définir des façons de pondérer les prédicats symboliques des règles heuristiques. Elles vont donc influencer fortement le choix des règles qui vont être utilisées pour étendre les roadmaps.

A chaque fois que l'une de ces actions est appliquée, un certain nombre de règles sera tiré aléatoirement et elles seront appliquées pour étendre les roadmaps.

Remarque :

• L'utilisation du contexte courant ne peut se faire que sur la prochaine action géométrique, après le contexte change et n'est plus utilisable. Ainsi l'action de *« recherche en largeur »* est spécialisée dans l'enrichissement des roadmaps avec contexte pour l'application directe des actions géométriques alors que l'action de *« recherche en profondeur »* essaie d'estimer la connexité des roadmaps pour corriger l'heuristique calculée.

• Une autre stratégie de modification des roadmaps qu'il pourrait être intéressant d'étudier, serait une stratégie d'oubli. Ce serait une action qui nettoierait les roadmaps en enlevant un certain nombre de nœuds et d'arcs qui n'apportent pas d'informations sur la connexité des roadmaps, mais qui ralentissent les algorithmes d'enrichissement de roadmaps ou de recherche de chemins faisables. Le coût heuristique d'une telle action sera alors inversement proportionnel aux coûts propres et heuristiques des actions d'enrichissement de roadmaps correspondantes.

6.4.3 La pondération des règles heuristiques.

En plus de leurs coûts, les *« actions d'apprentissage »* doivent aussi déterminer une pondération sur les actions géométriques qui ont des roadmaps à développer (6.5 *« pondération »*).

Grâce au fichier de liaison (§ 6.2.1) nous allons pouvoir associer les actions géométriques à un ensemble pondéré de règles heuristiques (§ 4.4) (R, P). Ces règles heuristiques doivent avoir une précondition du type *« Intérêt symbolique »* (§ 4.4.1).

Par exemple une action « `goto R P_R_TI P_R_TI_R-O_GP TI` » permet au robot R de se déplacer

vers une position pour prendre l'objet O. Il peut être intéressant d'associer à cette action des règles heuristiques portant par exemple sur l'enrichissement des roadmaps de :

- *Transit* (P=1) avec des méthodes générales ou spécifiques pour se relier à des nœuds correspondant à P_R_TI_R-O_GP (liés à la roadmap de *Grasp* ∩ *Placement*)
- *Grasp* ∩ *Placement* (P=0.5) en utilisant des méthodes qui créent des liens dans transit pour des positions d'objets existantes
- Roadmap de *Transit* relative (P=1) à l'objet O avec une méthode de diffusion.

Quand aSyMov applique une *« action d'apprentissage »* il va calculer la pondération des préconditions du type *« Intérêt symbolique »* de l'ensemble des règles :

`Mise à zéro des préconditions`
Pour Toute *action* pondérée par l'*action-apprentissage* **Faire**
  Pour Toute *précondition* symbolique correspondante à *action* **Faire**
    précondition.intérêt += P(*action*, *précondition*) × *action-apprentissage.pondération*[*action*]
  Fin Pour
Fin Pour
`Normalisation des préconditions`

Une fois les pondérations des préconditions symboliques calculées il est possible de choisir aléatoirement un certain nombre de règles qui vont elles même choisir un certain nombre de roadmaps à étendre et comment faire pour les étendre. Ce choix prendra en compte la pondération symbolique, mais aussi des informations géométriques avec les autres préconditions.

En plus de pondérer les règles heuristiques la précondition symbolique peut retourner un certain nombre de nœuds utilisables lors de l'enrichissement des roadmaps. Ces nœuds peuvent être utilisés comme critère d'arrêt de la méthode d'extension. Par exemple pour l'action « `goto R P_R_TI P_R_TI_R-O_GP TI` » l'apprentissage peut s'arrêter dès qu'il est possible d'atteindre, à partir de l'état géométrique courant, un nœud qui satisfait les contraintes de `P_R_TI_R-O_GP`.

Avec la pondération des règles heuristiques il est possible de choisir quelle roadmap étendre et avec quelle méthode, tout en prenant en compte le contexte géométrique de l'état courant (configuration des autres robots / objets) et en suivant des critères de choix à la fois géométriques et symboliques.

6.5 Le calcul des coûts heuristiques des actions.

L'heuristique des actions applicables est basée sur la longueur des plans trouvés par un planificateur symbolique (actuellement *« Metric-FF »* [Hoffmann 02]). Cette valeur est modifiée en tenant compte de la connectivité des roadmaps correspondant aux actions impliquées dans le plan. Si une action du plan heuristique n'implique pas une connectivité directe, c'est à dire s'il faut traverser plusieurs noeuds des GCEs pour atteindre les positions but, son coût heuristique sera augmenté. Pour les *« actions d'apprentissage »*, c'est l'inverse. Quand la connectivité est forte, le coût heuristique est augmenté.

Remarque :
Les problèmes qui nous intéressent ici ne sont pas des problèmes très complexes au niveau purement symbolique. Ce sont des problèmes où l'interaction symbolique/géométrique est forte.

Cela correspond à la grande majorité des problèmes de manipulation d'objets ou de réalisation de tâches. Ce seront des problèmes d'assemblage pour monter un meuble, des problèmes où le robot doit agir en tant qu'assistant dans une maison ou un bureau pour des tâches quotidiennes, dans cette maison il y aura tout au plus trois robots mobiles, ...

aSyMov n'est pas prévu pour résoudre de lourds problèmes de logistique ou des problèmes puzzle faisant intervenir des centaines d'objets et des dizaines de robots.

Dans ce cadre, le temps de calcul d'un plan par *« Metric-FF »* est faible[7] devant le temps de calcul des tests de collision nécessaires pour valider ou enrichir des roadmaps. Il est donc possible de relancer une planification symbolique complète quand c'est nécessaire.

Si les problèmes deviennent plus complexes il serait encore possible de chercher un plan dans un problème relaxé. Ainsi *« Metric-FF »* utilise comme heuristique le plan solution trouvé par un *« GraphePlan »* sur un problème relaxé (ne prenant pas en compte le retrait des faits par les actions). ASyMov utilise quand à lui *« Metric-FF »* comme heuristique (ne prend pas en compte la géométrie du problème). Pour des domaines plus complexes, il est possible d'utiliser directement l'heuristique de *« Metric-FF »*, c'est-à-dire le plan solution au problème relaxé.

L'algorithme de calcul des coûts heuristiques des actions (algorithme 3) est basé sur une propagation de connexité au niveau des nœuds accessibles des Graphes de Cinématiques Élémentaires (GCEs § 3.4) (`Correction_Plan_Heuristic` algorithme 4).

Nous allons voir cet algorithme sur l'exemple des deux chariots élévateurs développé pour les GCEs (figures 3.4 et 3.6) mais dans lequel on ne prend en compte qu'un seul chariot. Les figures 6.7 et 6.8 montrent le déroulement de l'algorithme de « `Correction_Plan_Heuristic()` » (4) sur un plan à 9 actions assez représentatif des actions possibles (figure 6.7.a). Il y a un certain nombre de positions spécifiques qui apparaissent dans ce plan et qui correspondent à des nœuds uniques des roadmaps (figure § 6.7.b).

Pour la première action le calcul des différents ensembles de nœuds est détaillé (figure 6.7).

Ainsi la fonction « `Nœuds_GCE_Pour_Position_Symbolique_Ajoutée` » détermine l'ensemble des positions symboliques ajoutées par l'action, c'est-à-dire l'ensemble des positions symboliques `P` qui sont présentes dans les effets de l'action par les prédicats `(on R P)`.

Pour chaque position symbolique P existe un certain nombre de nœuds $\mathcal{N}$ dans les roadmaps qui satisfont sa définition. $\mathcal{D}_g$ est l'ensemble des nœuds du GCE qui incluent les nœuds de $\mathcal{N}$.

La fonction « `Neuds_GCE_Pour_Action` » détermine l'ensemble des nœuds du GCE qui correspondent à l'action.

- Pour une action de déplacement géométrique « `goto R P1 P2` » cela représente l'ensemble des

[7] Le prétraitement est fait une seul fois, chaque appel à FF prend alors aux alentours de 0.3 seconds sur nos problèmes.

1: $min\_heuristique \leftarrow \infty$; $max\_heuristique \leftarrow 0$
$nb\_echec\_action \leftarrow 0$; $nb\_action \leftarrow 0$
2: **Pour Toute** *action-applicable* symbolique sur *état-sélectionné* **Faire**
3: nb_action += 1
4: $plan \leftarrow$ `Calcul_Plan`(*action-applicable*)
5: *action-apprentissage* ← *action-applicable*
6: *action-applicable.coût-heuristique* ← *plan.longueur*
7: *action-apprentissage.coût-heuristique* ← *plan.longueur*
8: *min-heuristique* ← *plan.longueur*
9: `Correction_Plan_Heuristic`()
10: **Si** ((*action-applicable.coût-heuristique* $< \infty$) et
(*max_heuristique* < *action-applicable.coût-heuristique*)) **Alors**
11: *max_heuristique* ← *action-applicable.coût-heuristique*
12: **Fin Si**
13: **Fin Pour**
14: *action-apprentissage-largeur.coût-heuristique* ←
$$\frac{min\_heuristic \times nb\_echec\_action + max\_heuristique \times (nb\_action - nb\_echec\_action)}{nb\_action}$$

Algorithme 3: Calcul du coût heuristique des actions

nœuds de $\mathcal{D}$ qui correspondent à la position symbolique `P1`.

- Pour une action de changement de roadmaps « `grasp R P1 O P2 R-O P3` » cela correspond à l'ensemble des nœuds du GCE qui sont reliés à des nœuds de $\mathcal{D}$ à travers des liens du GCE correspondant à ces changement de roadmaps.

Remarque :
Dans l'exemple $\mathcal{D}_g \cap \mathcal{D}_{action} \neq \emptyset$, ainsi la connexité de l'action est bonne. Si ça n'avait pas été le cas l'action aurait été forcément invalide et donc non sélectionnable par l'algorithme d'extension de l'espace d'état (§ 6.2.3).

La figure 6.8 montre le calcul de la correction du coût heuristique pour toutes les autres actions. Nous pouvons remarquer notamment les actions 4 et 8 qui ne sont pas applicables directement.

Dans l'état des connaissances actuelles de la topologie pour pouvoir les appliquer il faudrait auparavant appliquer d'autres actions.

Le changement de nœuds au niveau d'un GCE ne correspond pas généralement à une seule action. C'est pour cela que le nombre de pas *nb_pas* qui correspond en fait au nombre d'actions qu'il faudrait rajouter au plan symbolique pour pouvoir l'exécuter avec la connaissance topologique actuelle, correspond au nombre de changements de nœuds au niveau du GCE multiplié par le nombre moyen d'actions correspondant à un changement de nœuds dans le GCE.

1: $\mathcal{D} \leftarrow$ `Nœuds_GCE_Dans_Accessibilité_État`(*état-sélectionné*)
2: **Pour Toute** action A_i du *plan* **Faire**
3: $nb\_pas \leftarrow 0$
4: **Si** (`Est_Action_Géométrique`(A_i)) **Alors**
5: $\mathcal{D}_g \leftarrow$ `Nœuds_GCE_Pour_Position_Symbolique_Ajoutée`(A_i)
6: $\mathcal{D}_{action} \leftarrow$ `Nœuds_GCE_Pour_Action`($\mathcal{D}$, A_i)
7: **Si** ($\neg$ `Est_Connexe`($\mathcal{D}$, $\mathcal{D}_{action}$)) **Alors**
8: $nb\_pas \leftarrow \infty$
9: $\mathcal{D} \leftarrow \mathcal{D}_{action} \cup \mathcal{D}_g$
10: **Sinon**
11: **Tant Que** ($\mathcal{D}_{action} \cap \mathcal{D}_g = \emptyset$) **Faire**
12: nb_pas += *nb-actions-par-pas*
13: $\mathcal{D} \leftarrow$ `Diffuse_Arc_GCE`($\mathcal{D}$)
14: $\mathcal{D}_{action} \leftarrow$ `Nœuds_GCE_Pour_Action`($\mathcal{D}$, A_i)
15: **Fin Tant Que**
16: $\mathcal{D} \leftarrow \mathcal{D} \cap \mathcal{D}_{action}$
17: **Fin Si**
18: **Fin Si**
19: **Si** (($i = 0$) et (*nb-pas* > 0)) **Alors**
20: *action-applicable.coût-heuristique* $\leftarrow \infty$
21: nb_echec_action += 1
22: *action-apprentissage-largeur.pondération*[*action* $_i$] = nb_pas
23: **Sinon**
24: *action-apprentissage.pondération*[*action* $_i$] = $nb\_pas \times (diffusion\_ponderation)^i$
25: *action-apprentissage.coût-heuristique* += $\frac{cout\_max\_apprentissage \times (diffusion)^i}{nb\_pas+1}$
26: *action-applicable.coût-heuristique* += $\min(nb\_pas, cout\_max) \times (diffusion)^i$
27: **Fin Si**
28: **Fin Pour**

Algorithme 4: Correction du coût heuristique du plan : `Correction_Plan_Heuristic`()

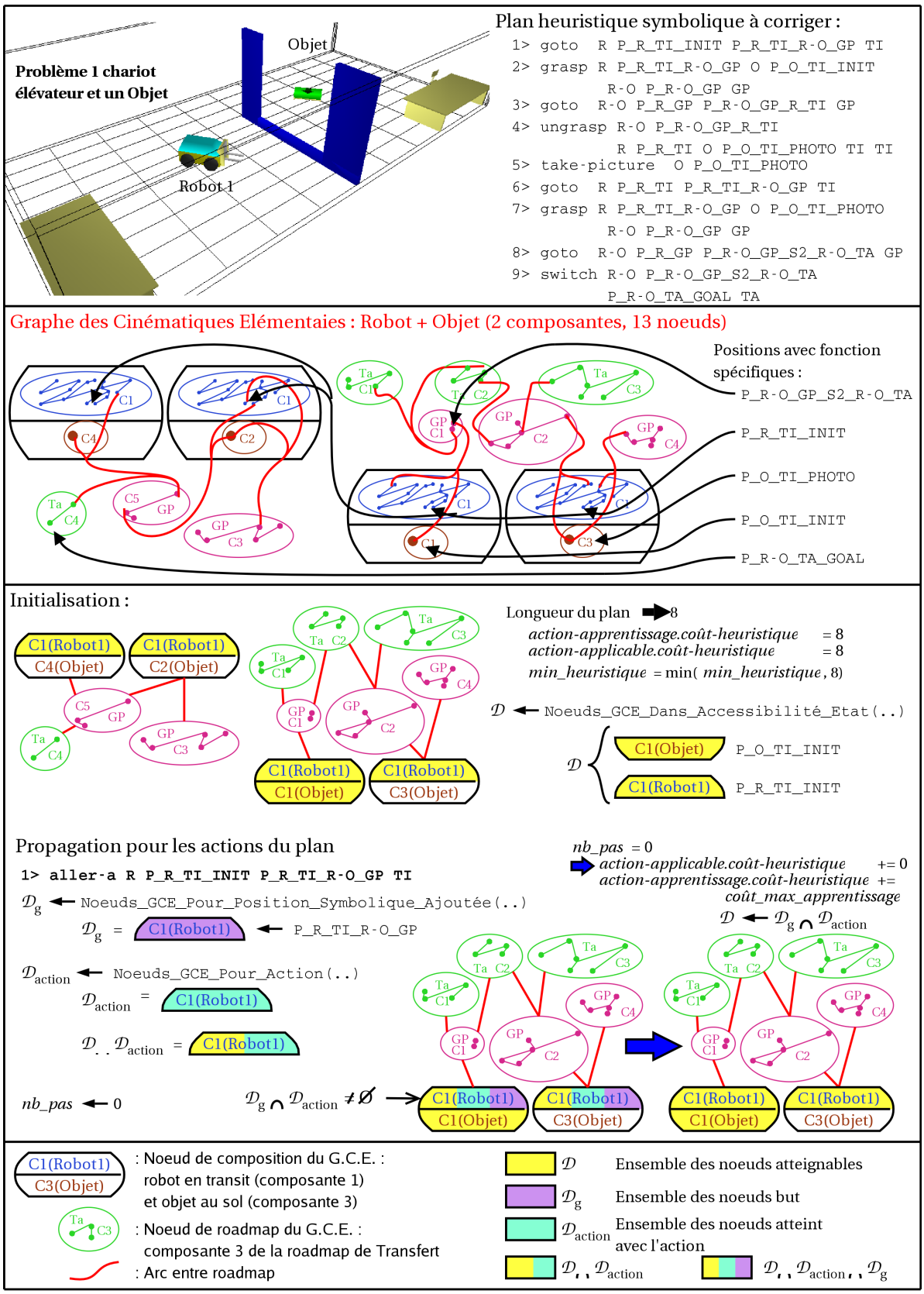


Figure 6.7: Exemple d'un robot chariot élévateur « R » qui doit porter un objet plat « O » à une position donnée pour prendre une photo, puis aller à une position spécifique de *Grasp* ∩ *Placement* pour enfin atteindre son but dans la roadmap de *Transfert*. Sur cet exemple est détaillé le calcul du coût heuristique des actions.

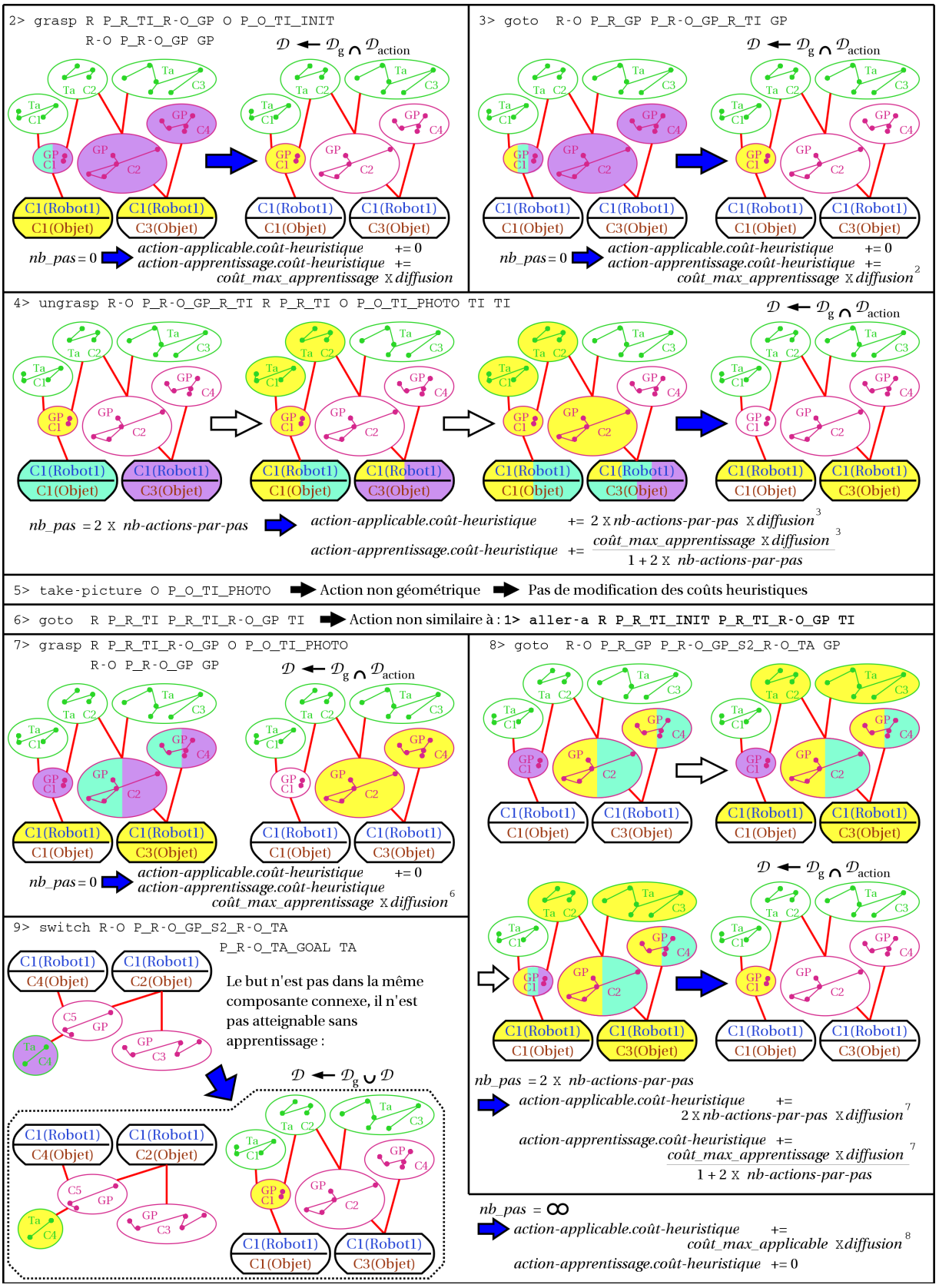


Figure 6.8: Suite et fin de l'exemple de la figure précédente.

Normalement un changement de nœuds du GCE correspond à deux actions : une action de déplacement pour atteindre les nœuds de liaison entre roadmaps et une action de changement de roadmaps. Ainsi nous pouvons estimer que *nb-actions-par-pas* vaut 2.

De plus comme nous l'avons dit, cette estimation se fait à partir des connaissances actuelles du GCE, un nouvel enrichissement de roadmaps pourrait créer de nouvelles connexions et diminuer ce nombre d'actions.

C'est pour représenter cette incertitude que l'on multiplie la correction par un facteur décroissant en fonction de la profondeur du plan. Plus l'action à réaliser sera éloignée, plus il y aura de chance que l'enrichissement des roadmaps modifie la topologie du problème créant de nouvelles connexités et réduisant ainsi le nombre d'actions à effectuer.

Cela tend à rendre l'heuristique un peu plus admissible afin de trouver des solutions optimales lors d'une recherche en largeur.

Par contre dans le cas où aSyMov fait une recherche à travers des roadmaps déjà apprises et sans actions d'enrichissement des roadmaps, la topologie ne va jamais varier au cours du temps. Dans ce cas le calcul de l'heuristique permet de comparer finement les actions avec la topologie du problème. Il faut alors avoir un facteur de $diffusion$ de 1 pour ne pas réduire l'influence de la topologie le long du plan.

Nous pouvons aussi remarquer sur la figure 6.8 que l'action 5 n'a pas d'effet sur les prédicats « `on` » et ne modifie donc pas les coûts heuristiques des actions.

L'action 9 met en évidence le cas où le but à atteindre se situe dans un nœud non présent dans les composantes connexes contenant $\mathcal{D}$. C'est ce cas qui nécessite le plus un enrichissement des roadmaps.

Notons par ailleurs que l'ensemble $\mathcal{D}$ peut regrouper plusieurs nœuds même s'ils appartiennent à des composantes connexes différentes. Cela évite de doubler le coût d'une non-connexité si l'on veut revenir dans la composante connexe précédente.

Remarque :
En plus du calcul de l'heuristique, aSyMov calcule pour les *« actions d'apprentissage »* des pondérations en fonction des actions. Elles déterminent l'intérêt qu'il peut y avoir à enrichir les roadmaps correspondant à ces actions § 6.4.3.

6.6 Les post-traitements.

A ce niveau, les plans trouvés par aSyMov ne sont pas optimaux ni même applicables directement car les chemins des roadmaps affinables doivent être transformés.

Le plan trouvé est composé d'un ensemble *« d'états d'aSyMov »* $\mathcal{E} = (\mathcal{E}_S, \mathcal{E}_A, \{\mathcal{E}_{G_1}, \ldots, \mathcal{E}_{G_n}\})$ (§ 6.1) liés entre eux par des actions valides. Ces *« états d'aSyMov »* contiennent des informations permettant de représenter toutes les instances possibles des *« états symboliques »* $\mathcal{E}_S$. Or ce qui nous intéresse ici, c'est uniquement les instances géométriques $\mathcal{E}_{G_i}$ ayant permis d'atteindre la solution.

Le post-traitement va avoir plusieurs étapes successives avant d'obtenir un plan solution :

1. Extraction de la suite $\mathcal{S} = ((\mathcal{E}_{0\,S}, \mathcal{E}_{0\,G_{k_0}}), \ldots, (\mathcal{E}_{n\,S}, \mathcal{E}_{n\,G_{k_n}}))$ correspondant au couple (*« état symbolique »*, *« état géométrique »*) des *« états d'aSyMov »* $\mathcal{E}_i$ permettant d'atteindre le but du problème.
2. Optimisation symbolique (§ 6.6.1).
3. Parallélisation symbolique (§ 6.6.2).
4. Post-traitement géométrique (§ 6.6.3).
5. Synchronisation des trajectoires (§ 6.6.3).

6.6.1 L'optimisation symbolique.

A cause du caractère aléatoire d'aSyMov, de l'estimation des heuristiques, de l'enrichissement des roadmaps qui peut ne pas trouver tous les chemins possibles, aSyMov n'est pas optimal, même si quand il fait une recherche en largeur il est proche de l'optimal.

Comme nous l'avons vu, les *« états symboliques »* peuvent correspondre à des *« états géométriques »* différents et par conséquent il existe plusieurs états dans aSyMov qui ont le même *« état symbolique »*. Parfois cela est justifié car cela correspond à des positions intermédiaires fondamentales (figure 6.2), et parfois cela ne l'est pas.

Nous allons pouvoir optimiser le plan donné par aSyMov en prenant en compte la particularité de ces états suivant les étapes :

1. Suppression des parties redondantes.
2. Transformation des positions spécifiques non utilisées en position générales.
3. Suppression des parties redondantes.
4. Suppression des parties de déplacement inutile.

6.6.1.1 Suppression des parties redondantes :

Il se peut que dans la suite des états qui mènent à la solution il existe deux *« états d'aSyMov »* $\mathcal{E}_i = (\mathcal{E}_{i\,S}, \mathcal{E}_{i\,A}, \{\mathcal{E}_{i\,G_1}, \ldots, \mathcal{E}_{i\,G_{k_i}}, \ldots, \mathcal{E}_{i\,G_{n_i}}\})$ et $\mathcal{E}_j = (\mathcal{E}_{j\,S}, \mathcal{E}_{j\,A}, \{\mathcal{E}_{j\,G_1}, \ldots, \mathcal{E}_{j\,G_{k_j}}, \ldots, \mathcal{E}_{j\,G_{n_j}}\})$ (i<j) qui possèdent à la fois le même *« état symbolique »* ($\mathcal{E}_{i\,S} = \mathcal{E}_{j\,S}$) et le même *« état géométrique »* ($\mathcal{E}_{i\,G_{k_i}} = \mathcal{E}_{j\,G_{k_j}}$) utilisé pour atteindre le but.

$\mathcal{E}_i$ et $\mathcal{E}_j$ peuvent être toutefois des états différents, ils peuvent avoir des *« états d'accessibilité »* $\mathcal{E}_A$ différents ou d'autres états géométriques qui ne sont pas communs aux deux états.

Toutefois dans ce cas, les deux états ont un rôle identique dans le plan trouvé. C'est pour cela que l'on peut retirer du plan $\mathcal{S}$ l'ensemble des couples $(\mathcal{E}_{i\,S}, \mathcal{E}_{i\,G_{k_i}})$, $(\mathcal{E}_{i+1\,S}, \mathcal{E}_{i+1\,G_{k_{i+1}}})$, …, $(\mathcal{E}_{j-1\,S}, \mathcal{E}_{j-1\,G_{k_{j-1}}})$ comme le montre la figure 6.9.

6.6.1.2 Transformation des « positions spécifiques » non utilisées en « positions générales » :

Une position symbolique `P` est dite *« spécifique »* si elle est utilisée dans des faits du type `(connection P ?p)` ou `(has-purpose P ?pos)`. Par exemple : `P_R_TI_INIT` avec `(has-purpose P INIT)`, `P_R_TI_R-O_TA`

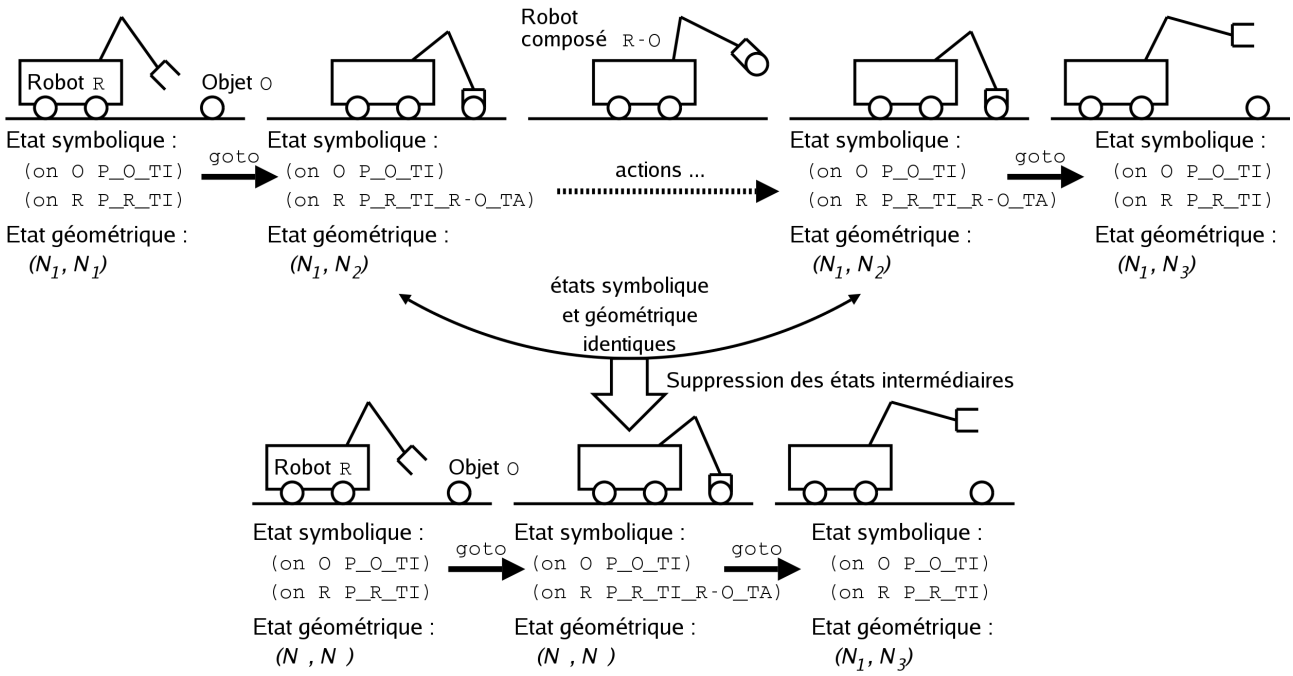


Figure 6.9: Exemple de suppression de partie de plan inutile par égalité des états symbolique et géométrique.

avec `(connection P_R_TI_R-O_TA P_R-O_TA)` sont des *« positions spécifiques »*.

Ces *« positions spécifiques »* vont représenter qu'une partie des noeuds des roadmaps et vont servir à accomplir des actions qui leur sont propres. Celles qui sont rattachées à des faits du type `(connection P ?P)` seront utilisées pour faire des changement de roadmaps. D'un autre coté, celles rattachées à des faits du type `(has-purpose P ?pos)` seront utilisées comme précondition d'actions incluant des contraintes de placement.

Une position symbolique `P` est dite *« générale »* si au contraire elle n'apparaît dans aucun fait du type `(connection P ?p)` [8] ou `(has-purpose P ?pos)`. Par exemple : `P_R_TI` ou `P_R-O_TA`.

Définition 9 *Si P1 est une* « position spécifique » *alors P2 est sa* « position générale associée » *si elle est une* « position générale » *et qu'elle représente le même robot et le même type de roadmap, i.e. elles ont les mêmes paramètres dans les faits belongs-to.*

Par exemple, `P_R_TI_INIT` est une position spécifique et `P_R_TI` est sa position générale associée à cause des faits `(belongs-to P_R_TI_INIT R TI)` et `(belongs-to P_R_TI_INIT R TI)` C'est-à-dire qu'il existe deux prédicats `(belongs-to P1 R TYPE)` et `(belongs-to P2 R TYPE)`.

Quand la fonction d'une *« position spécifique »* n'est pas utilisée, alors il est possible de la remplacer par une *« position générale associée »*. Ce qui est intéressant car une position générale est moins contrainte qu'une position spécifique.

Cela va permettre d'avoir des états symboliques identiques, et peut-être d'appliquer de nouveau l'algorithme de *« suppression des parties redondantes »*.

[8] Il faut noter qu'une position générale peut apparaître dans des faits du type `(connection ?p ?P)`, c'est à dire être la destination du lien. Il faut faire attention car ce prédicat est orienté.

Soit `P` une *« position spécifique »* avec `(belongs-to P R TYPE)`. Soit `Pg` son instance géométrique sur une partie du plan. Soient $\mathcal{E}_i$, ..., $\mathcal{E}_j$ les états d'aSyMov qui possèdent cette *position symbolique appliquée* `Pg`. Soit A_i, ..., A_{j-1} les actions appliquées à ces états ($\mathcal{E}_{i+1} = A_i(\mathcal{E}_i)$).

La *« position symbolique appliquée spécifique »* est utilisée pour sa fonction si une de ces conditions est vérifiée :

- `Pg` est une position but.
- Une des actions A_i, ..., A_{j-1} et même A_j si elle existe, utilise dans ses préconditions un fait du type `(connection P ?p)` ou `(has-purpose P ?pos)`.
  Typiquement, cela correspond à un changement de roadmaps (figure 6.10.a) ou à une action qui doit se faire en un lieu précis pour pouvoir communiquer avec une base fixe (figure 6.10.b).

La figure 6.10.c montre un cas où il est possible de remplacer dans l'état symbolique une position spécifique par une position générale sans remettre en cause l'intégrité du plan. Par contre, à ce niveau nous ne changeons pas l'état géométrique.

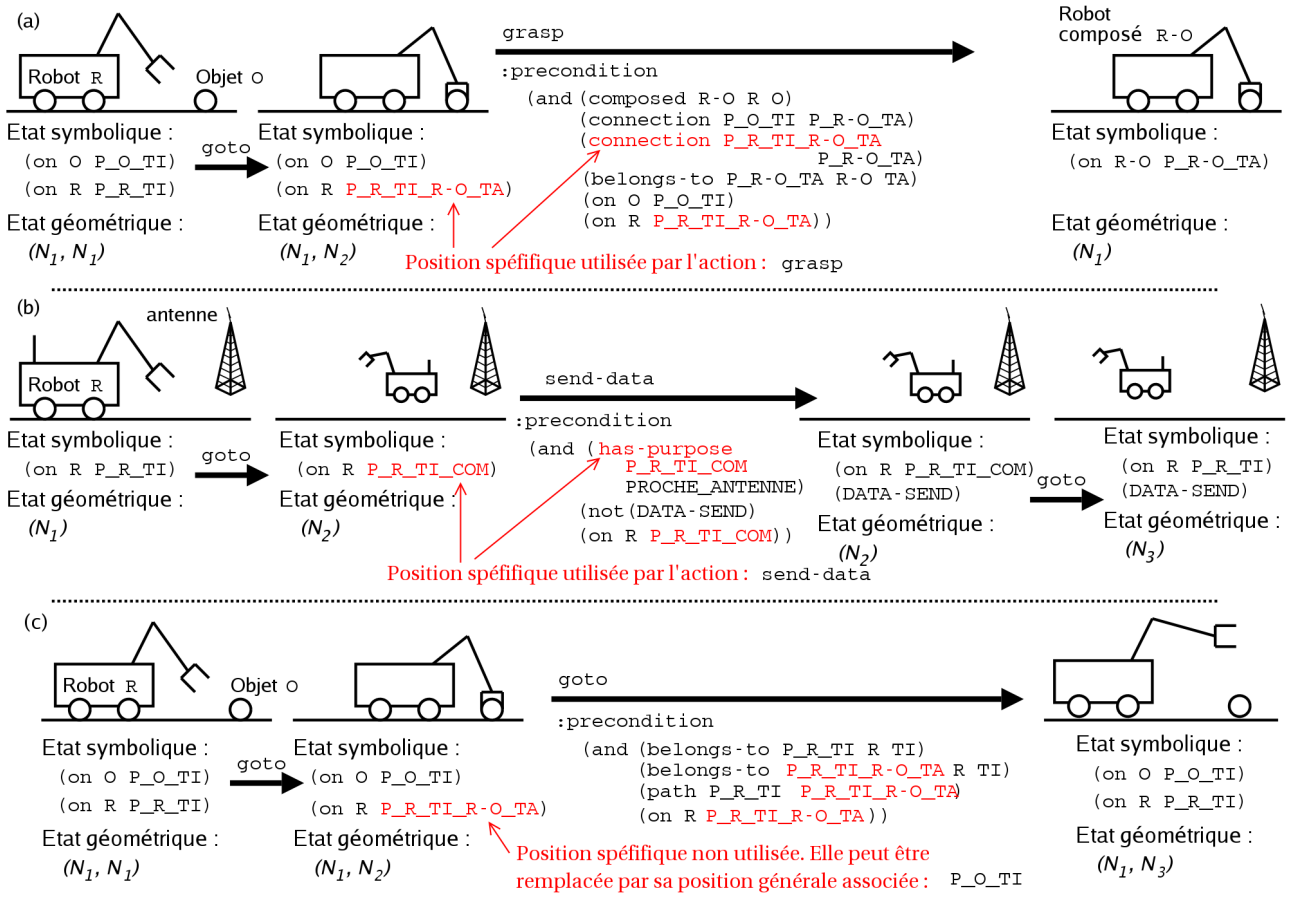


Figure 6.10: Exemple de positions spécifiques qui sont utilisées en tant que telles (a, b) et d'une position spécifique pouvant être remplacée par une position générale associée (c).

6.6.1.3 Suppression des parties de déplacement inutile.

Cette opération a pour but de modifier les états géométriques pour pouvoir avoir des états qui soient à la fois identiques au niveau symbolique et géométrique et ainsi enlever de nouveau des parties de plan d'une manière similaire à l'algorithme de *« suppression des parties redondantes »*. Par contre, cette opération va devoir vérifier s'il est toujours possible de trouver des trajectoires sans collisions et cela même en modifiant les états géométriques. C'est l'opération d'optimisation au niveau du plan qui est la plus coûteuse et est donc appliquée à un plan déjà en partie optimisé.

Pour cette opération nous recherchons donc des états symboliques identiques mais avec des états géométriques différents. Notons que grâce à l'opération 2 (Transformation des *« positions spécifiques »* non utilisées en *« positions générales »*) cela arrive assez souvent.

Soient $(\mathcal{E}_{i\,S}, \mathcal{E}_{i\,G_{k_i}})$ et $(\mathcal{E}_{j\,S}, \mathcal{E}_{j\,G_{k_j}})$ avec $i < j$ et tels que les états symboliques de $\mathcal{E}_{i\,S}$ et $\mathcal{E}_{j\,S}$ soient identiques. Nous allons alors essayer de remplacer $\mathcal{E}_{j\,G_{k_j}}$ par $\mathcal{E}_{j\,G_{k_i}}$. Pour cela aSyMov va appliquer toutes les actions à partir de $\mathcal{E}_j$ jusqu'à la fin du plan en utilisant l'état géométrique $\mathcal{E}_{i\,G_{k_i}}$. ASyMov va ainsi recréer une nouvelle suite d'états géométriques $\mathcal{E}_{j\,G_{k'_j}}$, …, $\mathcal{E}_{n\,G_{k'_n}}$ tel que $\mathcal{E}_{j\,G_{k'_j}} = \mathcal{E}_{i\,G_{k_i}}$. S'il n'arrive pas à recréer cette suite d'états géométriques c'est qu'une partie des opérations qui ont permis de mener aSyMov à l'état géométrique $\mathcal{E}_{j\,G_{k_j}}$ est indispensables à la réalisation du plan (figure 6.11).

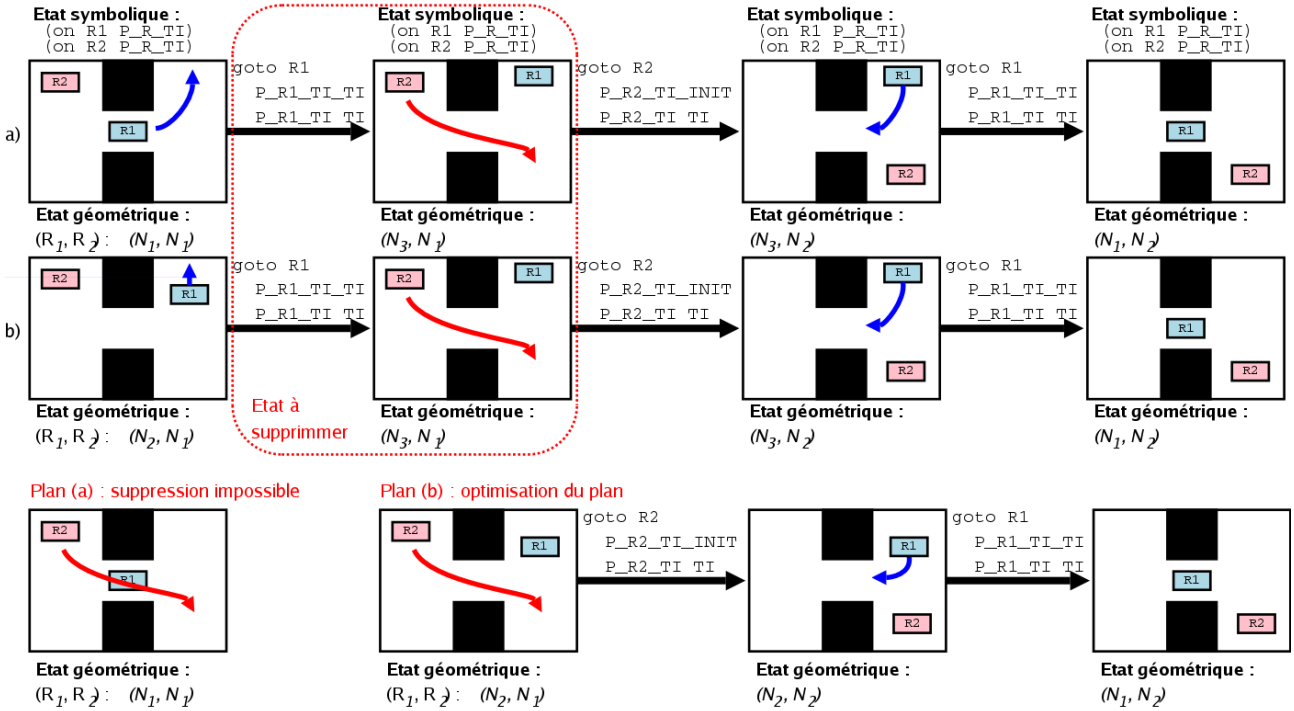


Figure 6.11: Exemple d'un même plan avec deux états géométriques initiaux différents. Dans un cas l'état géométrique peut être reporté (b), il y a un gain d'une action, nous avons de nouveaux états géométriques. Dans l'autre cas (c) la position intermédiaire est indispensable. Seul le noeud 3 du robot R1 n'est ni une position initiale, ni un but, c'est le seul qui peut être modifié.

Remarque :

Il est encore parfois possible d'optimiser le plan si nous nous plaçons dans l'espace des plans et pas dans l'espace d'état. En effet, il suffit qu'un prédicat de l'état soit différent pour que l'optimisation ne soit pas possible. Par exemple, si le contenu d'un réservoir d'un robot R1 varie avec l'action A_i, les états $\mathcal{E}_{i+1}$, ..., $\mathcal{E}_n$ qui suivent cette variations seront différents au niveau symbolique des états $\mathcal{E}_0$, ..., $\mathcal{E}_i$. Ainsi les mouvements des autres robots ne pourront être supprimés entre des états $\mathcal{E}_k$ ($k \leq i$) et $\mathcal{E}_l$ ($l > i$). L'espace d'état ne permet pas de faire apparaître l'indépendance des robots.

Si nous voulons pousser plus loin encore l'optimisation des plans il faut alors travailler dans l'espace des plans et avoir une représentation partiellement ordonnée des actions. Cela peut être fait après l'étape de parallélisation qui va suivre.

6.6.2 Parallélisation du plan.

Pour cela nous utilisons une technique basée sur une table triangulaire [Régnier 92].

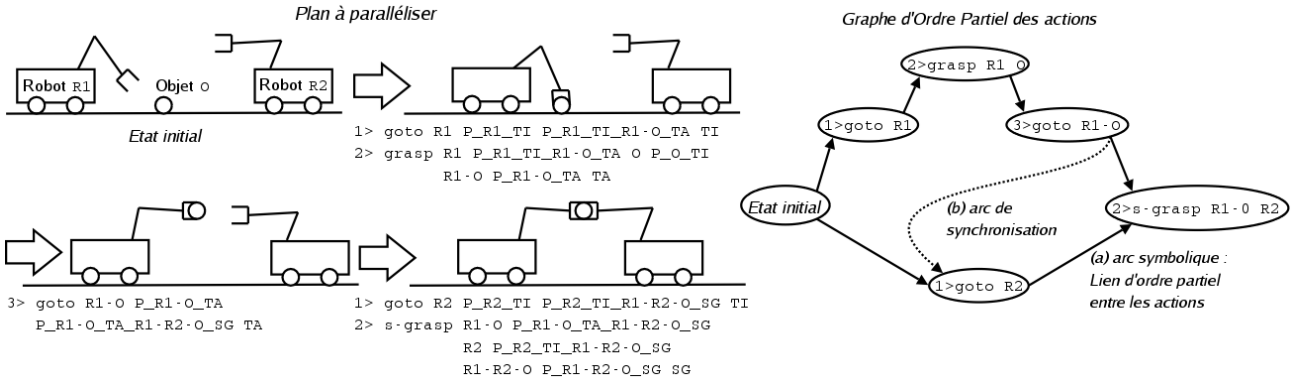


Figure 6.12: Représentation d'un plan parallélisé au niveau symbolique. Les contraintes d'ordre dues aux préconditions symboliques sont représentées par des arcs symboliques (a). Les contraintes qui assurent une faisabilité géométrique sont représentées par des arcs de synchronisations (b) (ordre séquentiel à ce niveau de l'algorithme).

La construction d'une table triangulaire [R.E. Fikes 72] permet de représenter les dépendances entre actions. Pour un plan de n actions, la table est représentée par la partie triangulaire inférieure gauche d'un tableau de taille $(n+1)$ (table 6.1).

La colonne 0 représente les faits de l'état initial. La diagonale représente les actions du plan séquentiel. La colonne C_i représente les faits ajoutées par l'action A_i alors que la ligne L_i représentent ses préconditions. Ainsi la case $C_{i\,j}$ représente les faits ajoutés par l'action A_i qui servent de précondition à l'action A_j.

La ligne $n+1$ représente les faits de l'état final provenant de l'état initial (colonne 0) ou ajoutés par les actions A_i (colonne i).

Une fois cette table construite, il est possible de construire un *« graphe d'ordre partiel »* entre les actions [Régnier 92] (figure 6.12). Deux actions A_i et A_j seront ordonnées si et seulement si au moins l'une des conditions suivantes est vérifiée :

	0	1	2	3	4	5
1	(on R1 P_R1_TI)	`1> goto R1`				
2	(on O P_O_TI)	(on R1 P_R1_TI_R1-O_TA)	`2> grasp R1 O`			
3			(on R1-O P_R1-O_TA)	`3> goto R1-O`		
4	(on R2 P_R2_TI)				`4> goto R2`	
5				(on R1-O P_R1-O_TA_R1-R2-O_SG)	(on R2 P_R2_TI)	`5> s-grasp R1-O R2`
6						(on R1-R2-O P_R1-R2-O_SG)

Table 6.1: Table triangulaire des actions du plan de la figure 6.12. Seuls les faits non statiques sont représentés ici.

- la case $C_{i\,j}$ n'est pas vide;
- les ensembles des préconditions de A_i et des effets de A_j sont incompatibles;
- il existe au moins une suite d'actions (A_i, ..., A_k, ..., A_j) telle que tout couple d'actions successives de cette suite soit ordonnée.

Toutefois, il n'est pas possible d'appliquer directement cette parallélisation puisqu'elle ne prend pas en compte les contraintes géométriques. Il se peut que deux actions de déplacement symboliquement indépendantes ne soient plus valides géométriquement si leur ordre est inversé. La figure 6.11.a montre clairement un cas où deux actions a priori indépendantes au niveau symbolique ne peuvent pas être parallélisées.

Le plan solution sera représenté par un graphe orienté possédant deux types d'arc :

- Les *« arcs symboliques »* (figure 6.12.a) représentent l'ordre partiel des actions trouvé par l'utilisation de la table triangulaire.
  Ces arcs devront être toujours respectés par les techniques d'optimisation géométrique.
- Les *« arcs de synchronisation »* (figure 6.12.b) sont utilisés pour synchroniser les trajectoires. Ils peuvent donc s'appliquer au milieu d'une action (pour les actions de déplacement).
  A ce niveau ils représenteront l'ordre total du plan séquentiel des actions trouvées par aSyMov. Ils seront modifiés par des opérations de synchronisation des trajectoires géométriques.

6.6.3 Post-traitement géométrique

Le post-traitement géométrique consiste en quatre étapes :

1. Extraction des trajectoires.
2. Optimisation des trajectoires (§ 2.2.3).
3. Transformation des trajectoires affinables (§ 2.8.6).
4. Synchronisation des trajectoires.

6.6.3.1 L'extraction des trajectoires :

Contrairement aux techniques de validation des plans (§ 6.3.1), cette extraction de trajectoire se fait sur la base d'un A* [Nilson 80] pour trouver la trajectoire la plus courte dans la roadmap.

Remarque :
Cette opération est étonnamment longue. Pour les exemples les plus simples elle peut prendre presque la moitié du temps de recherche du planificateur. Ceci permet de mettre en évidence les performances de la fonction « `Valide` » de recherche d'existence de chemins (qui n'assure pas l'optimalité) § 6.3.1. Celle-ci peut être appelée des dizaines de fois à chaque validation d'une action géométrique.

6.6.3.2 L'optimisation des trajectoires :

Elle est basée sur un algorithme de tirage aléatoire et une bande élastique (§ 2.2.3).

Remarque :
Il est intéressant d'effectuer cette optimisation à ce niveau puisque le plan a encore une exécution séquentielle. Ainsi chaque trajectoire peut être optimisée dans le contexte de l'état précédent l'action. De plus cette optimisation n'aura aucune conséquence sur les actions suivantes puisque celles-ci attendent la fin de l'action et donc l'arrivée en position finale de la trajectoire qui ne sera pas déplacée.

6.6.3.3 La transformation des trajectoires :

La roadmap de *Grasp* ∩ *Placement* possède une propriété de réduction et un chemin dans *Grasp* ∩ *Placement* va être transformé en une succession de chemin de *Transit* et de *Transfert*.

La méthode de transformation est celle développée par Anis Sahbani (§ 2.8.6).

Remarque :
De même que l'optimisation de trajectoire, il est intéressant d'effectuer cette opération avant la parallélisation des plans pour avoir un contexte fixé pendant cette opération.

6.6.3.4 La synchronisation de trajectoires.

C'est la dernière étape du post-traitement. Elle n'a malheureusement pas été implémentée dans la version actuelle d'aSyMov. Par contre nous avons démontré la faisabilité de cette étape.

Elle sera basée sur une technique *« d'insertion de plan »* avec priorité dynamique que nous avons développée pour la coordination de plusieurs robots autonomes [Gravot 01].

Les travaux de Stéphane Cambon [9] développeront plus en détail ce point précis de l'algorithme.

Nous avons présenté ici toutes les étapes du planificateur *« aSyMov »*. Nous allons maintenant étudier les résultats obtenus par son implémentation dans le chapitre suivant.

[9] *« aSyMov »* a été développé avec la collaboration de Stéphane Cambon dont la thèse détaillera certains points rapidement évoqués dans cette thèse.

Chapitre 7

Résultats

Dans ce chapitre nous allons présenter les résultats qu'a obtenu aSyMov sur trois domaines différents. Nous en profiterons pour faire quelques remarques sur les mécanismes internes d'aSyMov et sur les pistes à suivre pour améliorer encore ses performances.

Ces trois domaines sont : le domaine des chariots élévateurs, le domaine des tours de Hanoi et le domaine *« IKEA »*.

Le domaine des chariots élévateurs est plutôt simple géométriquement, mais il nous a permis de faire apparaître, suivant les problèmes traités, tous les subtilités d'aSyMov.

Le domaine des tours de Hanoi met en évidence l'utilisation de notions symboliques au sein d'un planificateur de manipulation. Les différents problèmes étudiés sur ce domaine nous ont permis de mettre en évidence l'influence de l'aspect géométrique de l'environnement sur les plans obtenus.

Enfin le domaine IKEA met en scène un problème d'assemblage *« réaliste »* mêlant à la fois des informations symboliques et géométriques.

7.1 Le domaine des chariots élévateurs.

Pour ce domaine nous avons un environnement relativement simple (figure 7.1.a) ainsi que deux chariots élévateurs qui doivent transporter un objet plat d'un bout à l'autre de la pièce.

Une particularité de ces chariots élévateurs est qu'ils ne peuvent saisir l'objet que d'un seul côté (figure 7.1.b). La prise est définie de manière continue, les robots ont une infinité de possibilités pour saisir l'objet (figure 7.1.c).

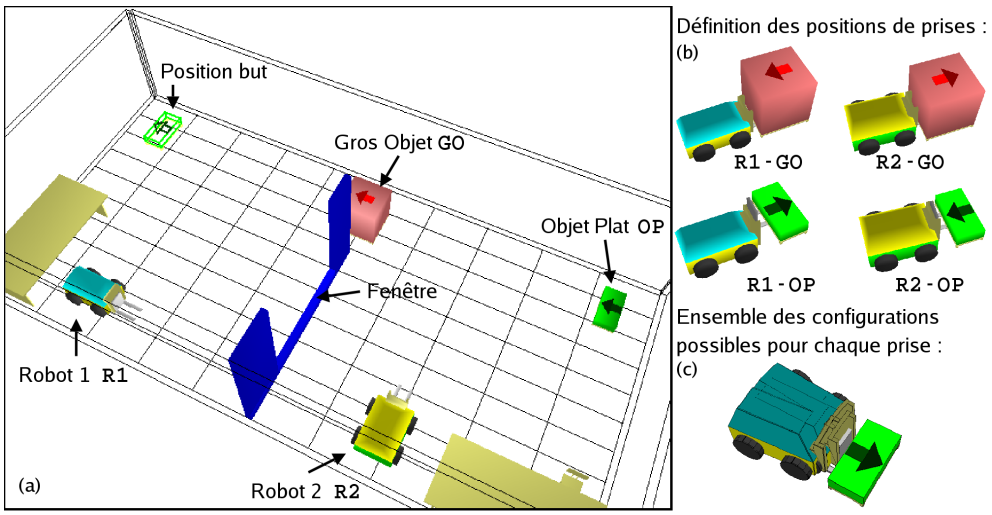


Figure 7.1: Domaine des chariots élévateurs. Le but est de déplacer l'objet plat dans sa position finale.

Vues les positions initiales et finales de l'objet, seul le robot `R1` peut le saisir en position initiale et seul le robot `R2` peut le mettre en position finale (figure 7.2.a).

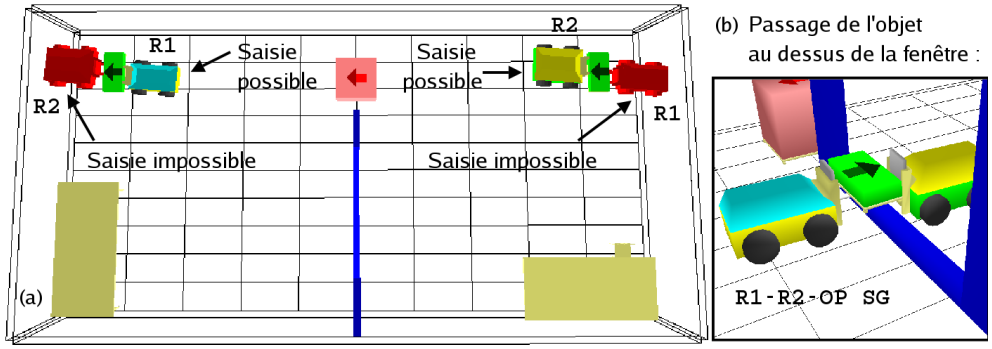


Figure 7.2: Spécificité du problème. (a) Robots pouvant prendre l'objet plan dans sa position initiale et finale. (b) Passage de l'objet directement d'un robot à l'autre par une prise simultanée `SG`. Il se forme alors un robot `R1-R2-OP` composé des deux chariots élévateur et de l'objet plat.

Il faut donc que les robots puissent se passer l'objet en question. Pour cela, ils peuvent soit le poser au sol, soit se le passer directement d'un robot à l'autre à travers la fenêtre (figure 7.2.b).

Nous avons défini plusieurs variantes. La première variante $\mathcal{P}_1$ augmente les difficultés géométriques en ajoutant un gros objet `GO` déplaçable qui va bloquer le passage (figure 7.1.a). Il faudra alors le mettre dans une position intermédiaire non gênante pour pouvoir accomplir le but.

Ce problème possèdent deux catégories de solutions : celle qui passe l'objet par la fenêtre (figure 7.4) et celle qui déplace le gros objet (figure 7.3).

Nous avons déjà détaillé la seconde solution dans le chapitre précédent (§ 6.3), pour illustrer le processus de validation qui va propager les contraintes géométriques afin de trouver une position intermédiaire adéquate.

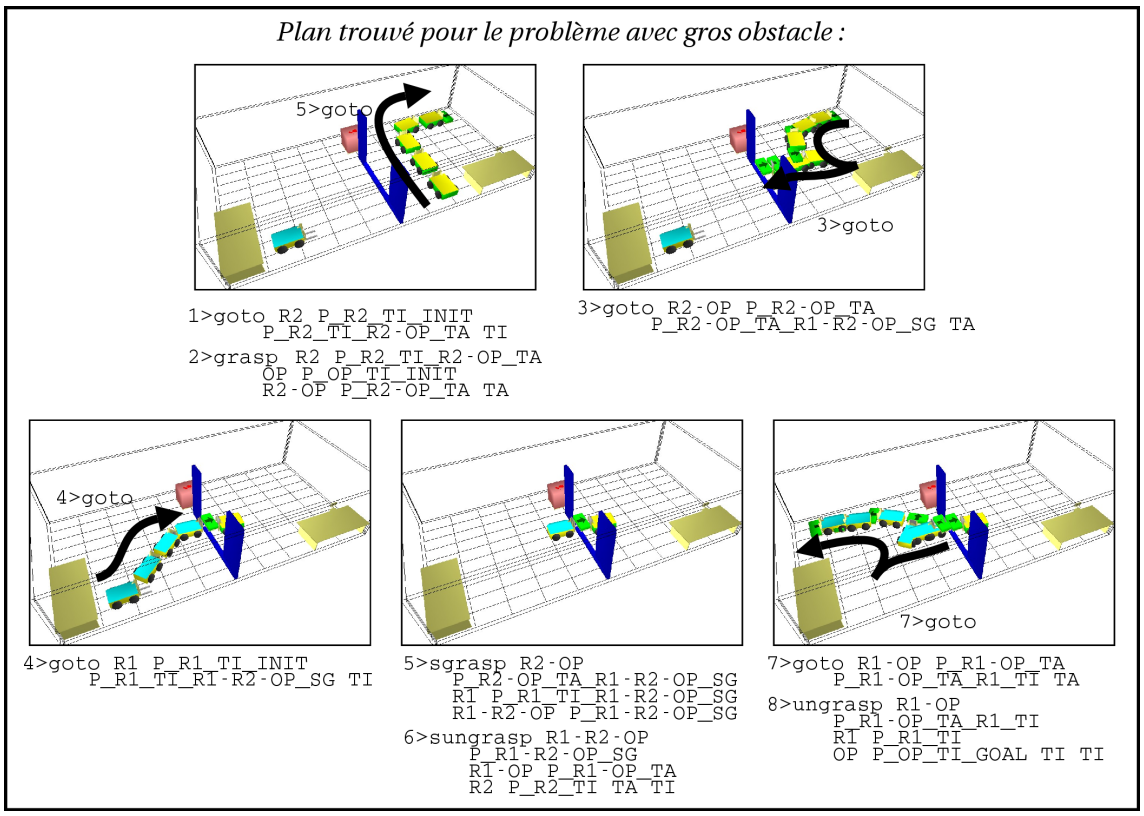


Figure 7.3: Première catégorie de solution au problème $\mathcal{P}_1$. L'objet est passé au dessus de la fenêtre. Il y a une saisie simultanée `SG` de l'objet par les deux chariots élévateurs pour former un nouveau robot (action 4).

Le déplacement du gros objet induit 4 actions supplémentaires (action 5 à 9) qui rendent le plan solution plus long que celui de la solution de la figure 7.3. Ces 4 actions supplémentaires ont été nécessaires pour prendre en compte les collisions entre robots. Or celles-ci ne sont prises en compte ni au niveau du plan symbolique, ni au niveau des GCEs.

Il en résulte que plan heuristique qui a guidé le plan solution de la figure 7.4 est plus court de 4 actions que le plan réel. Ce plan heuristique est donc plus avantageux que celui qui permet d'avoir la première catégorie de solution (figure 7.4). Il est alors naturel de constater dans les résultats une prépondérance du plan consistant à déplacer le gros objet, même si le coût réel de ce plan est supérieur à la solution qui passe l'objet par la fenêtre.

Une seconde variante $\mathcal{P}_2$, met à la place de l'objet `GO` une porte automatique qui ne peut être ouverte que si l'on place une clé magnétique à une position précise (figure 7.5). Cette clé magnétique est purement virtuelle et donc n'apparaîtra qu'au niveau symbolique. Le robot `R2` devra aller la chercher puis ouvrir la porte pour laisser le passage à l'autre robot afin qu'ils puissent s'échanger l'objet plat (figure 7.6).

Pour cette variante nous avons dû séparer symboliquement les deux pièces (`ROOM_1`, `ROOM_2` figure 7.5). Nous avons deux actions de déplacement. Une action « `goto` » qui en plus de ses préconditions classiques va demander que les deux positions symboliques appartiennent à la même pièce. L'autre

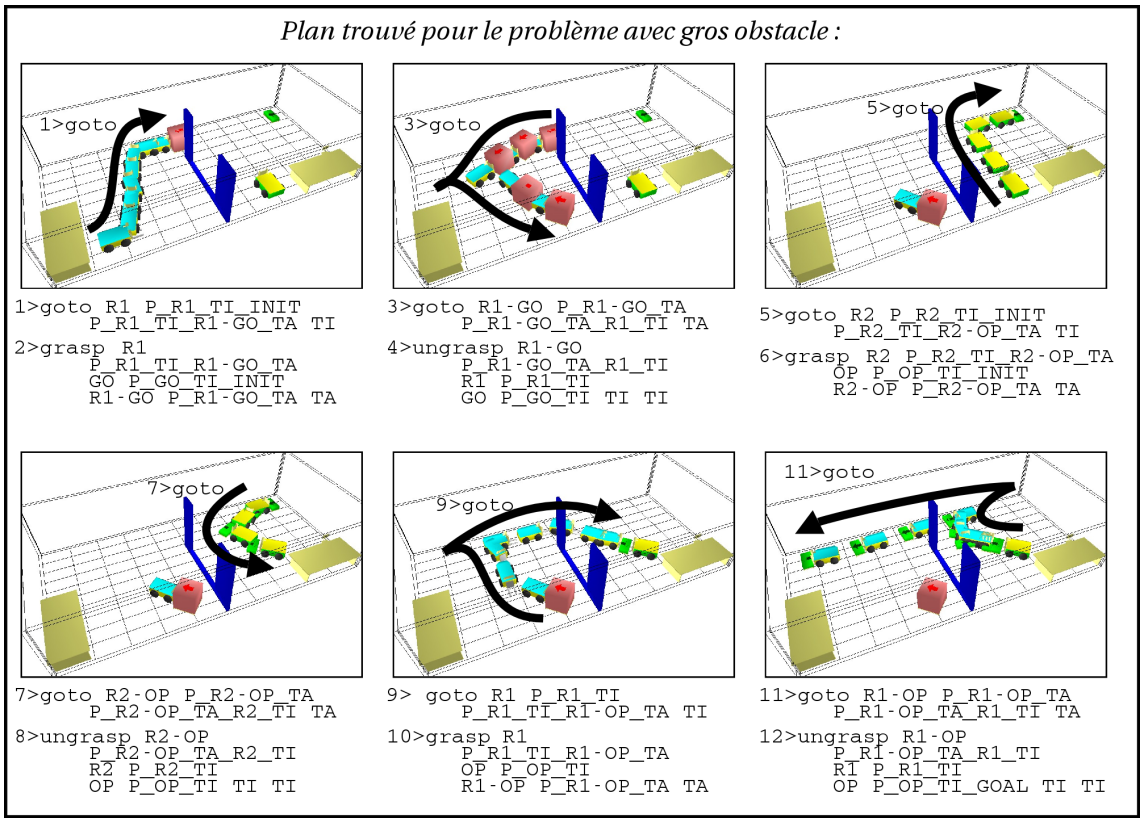
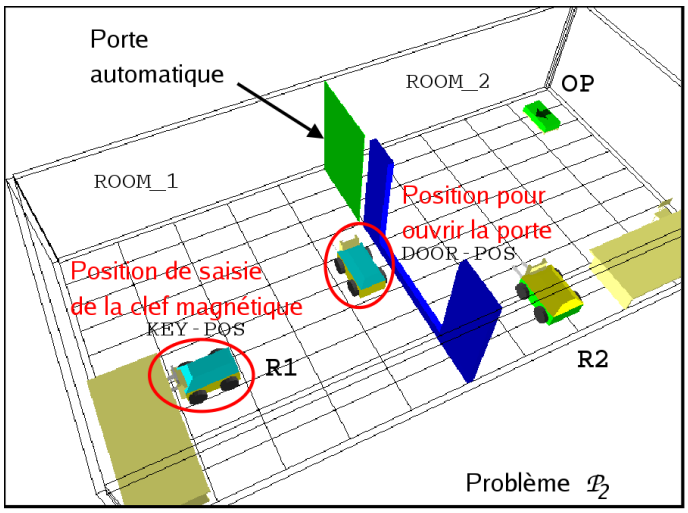


Figure 7.4: Seconde catégorie de solution au problème $\mathcal{P}_1$. Le gros objet est déplacé afin de trouver un chemin pour que le robot `R2` puisse aller prendre l'objet plat.



Faits supplémentaires :

```
(KEY-ON-TABLE)
(KEY-ON-ROBOT ?r - robot)
(KEY-IN-DOOR)
(DOOR-OPEN)
```

Constantes supplémentaires :

```
robot : DOOR
has-purpose : ROOM_1
              ROOM_2
              KEY-POS
              DOOR-POS
```

Actions supplémentaires :

```
get-key
open-door
move-door
goto-through-door
```

Figure 7.5: Variante $\mathcal{P}_2$ du problème avec une porte qui ne peut s'ouvrir qu'avec une clef magnétique placée sur le bureau. La porte est considérer comme un robot. Il possède une action de déplacement spécifique « `move-door` » pour actionner son ouverture. Celle-ci ne sera effectivement possible qu'une fois qu'un robot aura mit la clef dans la porte « `open-door` ».

action « `goto-through-door` » imposera que les deux positions ne soient pas dans la même pièce et que la porte soit ouverte (`DOOR-OPEN`). Grâce à ces informations le guidage heuristique est beaucoup plus performant et il se trouve que ce problème est résolu plus facilement que le domaine basique.

Pour $\mathcal{P}_2$ nous avons aussi supprimé la possibilité de passer `OP` par la fenêtre afin d'avoir seulement la solution qui implique l'ouverture de la porte.

Il faut remarquer que la porte est mobile. Pour cette raison, nous l'avons représentée comme un robot autonome qui pourra se déplacer dans sa position ouverte (action `move-door`) une fois que la clef sera mise dans la porte (le fait `(KEY-IN-DOOR)` effet de l'action `open-door`).

Les objets « *robots* » d'aSyMov peuvent ainsi servir à représenter divers mécanismes.

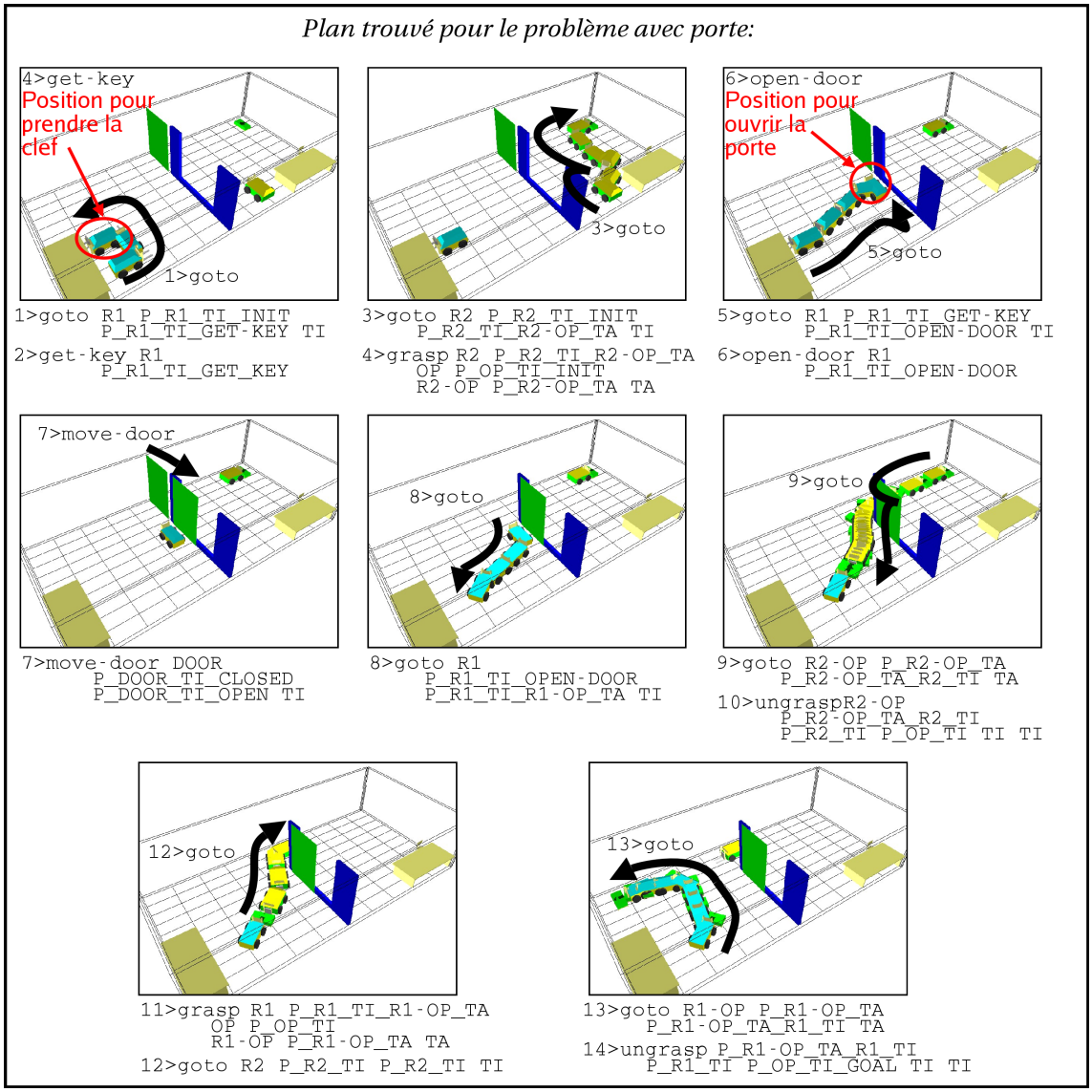


Figure 7.6: Solution du problème $\mathcal{P}_2$ avec la porte magnétique.

Remarque :

La représentation des actions dans aSyMov est très détaillée. Pour pouvoir faire une opération de

déplacement d'un objet il faut au moins 4 actions : une action (goto) de déplacement du robot pour venir saisir l'objet, une action (grasp) pour le prendre, une action (goto) de transfert et enfin une action (ungrasp) pour le déposer (figure 7.4 actions 1 à 4, 5 à 8 ou 9 à 12).

Il pourrait sembler évident de regrouper ces actions au moins deux par deux. Pourtant il n'est pas possible de le faire systématiquement.

Prenons l'exemple de la figure 7.3 : un robot `R2` va passer un objet `OP` à un autre robot `R1` par l'intermédiaire d'une saisie simultanée (sans le poser sur le sol). Dans ce cas il faut aussi 4 actions, mais elles ne sont pas regroupées de la même façon : une action de transfert du robot composé `R2-OP` pour amener l'objet dans une position de saisie simultanée (figure 7.3 action 3), une action de transit de `R1` pour aller dans cette même position (figure 7.3 action 4), une action de saisie simultanée pour former le robot `R1-R2-O` (figure 7.3 action 5) et une action de séparation pour avoir le robot `R1-O` (figure 7.3 action 6).

Cette fois-ci il faut déplacer les deux robots avant de faire l'échange. ASyMov doit être capable de travailler avec les prédicats élémentaires que nous avons établis ce qui lui permet d'avoir une grande souplesse dans la définition de ses actions.

Mais cette représentation pourrait bénéficier pleinement des techniques des planificateurs domaine dépendant [Erol 95, Bacchus 96]. En effet la manipulation demande d'effectuer les actions décrites plus haut dans un ordre précis. Comme cet ordre n'est pas donné, et que les actions élémentaires sont assez fines, la complexité du problème est augmentée.

Remarque :
Le plan trouvé dans la figure 7.6 n'est pas optimal en nombre d'actions, mais toutes les actions du plan sont nécessaires avec la connaissance de la topologie acquise par aSyMov au cours de la planification.

Nous pouvons constater que l'action 12 consiste simplement à déplacer le robot `R2` pour qu'il ne gène pas le robot `R1`. Avec une meilleur connaissance de la topologie, il aurait été possible de trouver un chemin pour `R1-OP` même sans déplacer le robot `R2`.

ASyMov ne peut pas garantir de trouver une solution avec le nombre minimal d'actions, même après optimisation. Il est limité par la connaissance qu'il a de la topologie du problème.

7.2 Le domaine des tours de Hanoi.

Les tours de Hanoi sont un problème classique de la planification de tâches. Il y a trois tours possibles, et un objet ne peut être placé que sur un objet plus gros que lui. Il faut alors passer d'une configuration des tours à une configuration but. Cela nécessite souvent plusieurs opérations intermédiaires (figure 7.7).

Dans notre cas nous allons ajouter à ce problème un robot qui va déplacer les objets et nous allons donner une consistance « physique » aux objets et à l'environnement. Ceci sera notre problème *« basique »* $\mathcal{P}_0$ des tours de Hanoi (figure 7.8.a). De plus le chariot qui va déplacer ces tours sera non-holonome. Il utilisera une méthode locale du type *« Reeds & Shepp »* (succession de lignes droite

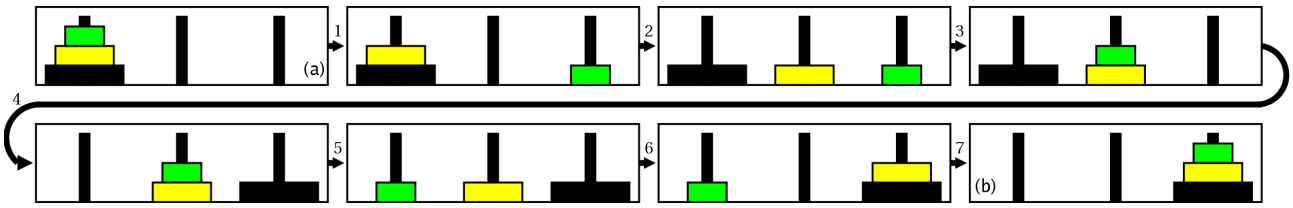


Figure 7.7: Plan solution du problème des tours de Hanoi de la planification de tâches pour passer de l'état initial (a) à l'état final (b).

et d'arcs de cercles de rayon fixé).

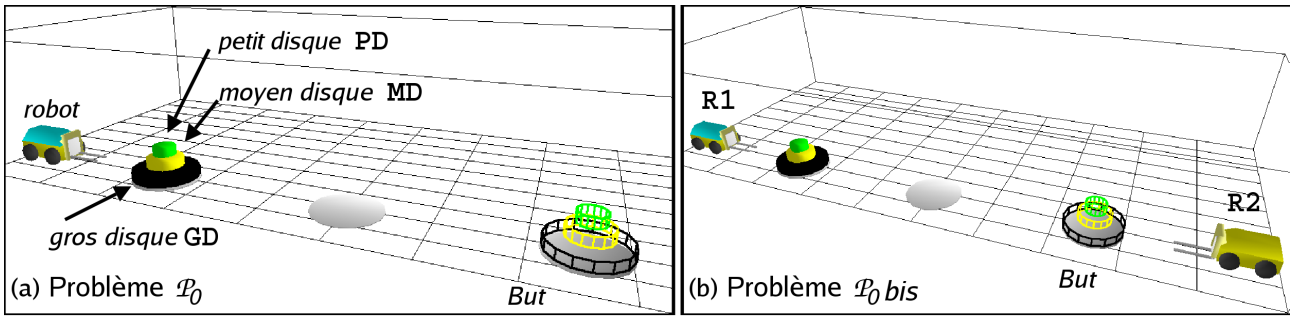


Figure 7.8: Problème géométrique des tours de Hanoi. Le but fixé est celui présenté dans la figure 7.7. Le problème $\mathcal{P}_0$ (a) consiste a déplacer les disques à l'aide d'un chariot élévateur. Sa variante $\mathcal{P}_0 bis$ (b) possède deux chariots élévateurs.

La première variante sur ce domaine est le problème $\mathcal{P}_0 bis$ (figure 7.8.b). Il utilise deux chariots élévateurs. Dans ce problème la taille de l'espace d'état et les actions possibles pour chaque état augmentent de façon importante.

Nous allons aussi définir plusieurs variantes du problème pour montrer l'influence de l'aspect géométrique sur la génération de plan. Pour cela nous avons quatre problèmes $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ et $\mathcal{P}_4$ de difficulté croissante au niveau géométrique en fonction de la taille de l'obstacle dans l'environnement (figure 7.9.a, b, c et d).

Pour les problèmes $\mathcal{P}_3$ et $\mathcal{P}_4$ (figure 7.9.b et c), il est impossible au chariot élévateur de transporter le gros disque en passant par le haut de l'environnement, de plus il est impossible de passer à côté du disque moyen (figure 7.10).

Pour être exact, à la conception des problèmes, nous pensions que dans tous les cas de figure, il était impossible de passer le gros disque en haut de l'environnement. Il se trouve, qu'à cause de la méthode locale utilisée, ce passage est possible pour les problèmes $\mathcal{P}_1$ et $\mathcal{P}_2$. Cela nous à permis de voir que même sur un environnement aussi simple, il peut être parfois difficile pour l'utilisateur de connaître tous les chemins possibles.

Le problème $\mathcal{P}_3$ est identique au problème $\mathcal{P}_1$ hormis les murs extérieurs qui ont été rajoutés pour interdire le passage du gros objet par le haut de l'environnement.

La largeur de l'obstacle du problème $\mathcal{P}_2$ est telle qu'elle interdit le passage du gros disque. Mais il contraint encore plus l'environnement. Il est alors très difficile de trouver un chemin qui permet au

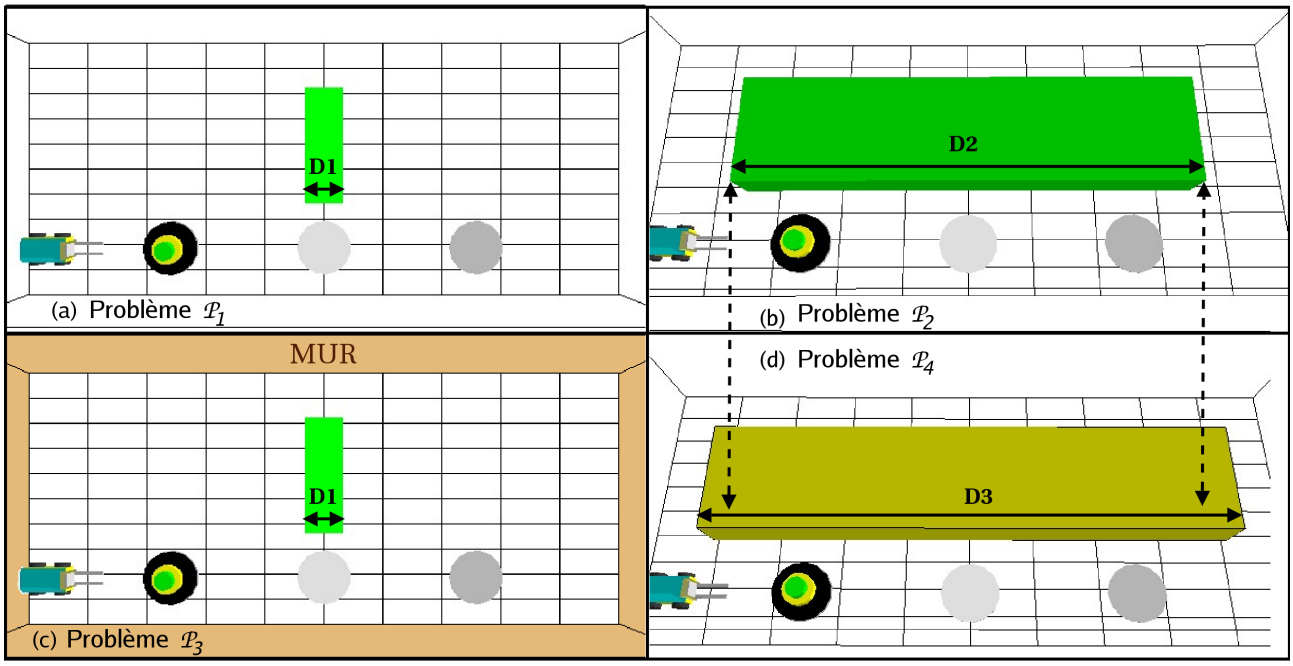


Figure 7.9: Variantes du problèmes des tours de Hanoi avec l'ajout d'un obstacle statique.

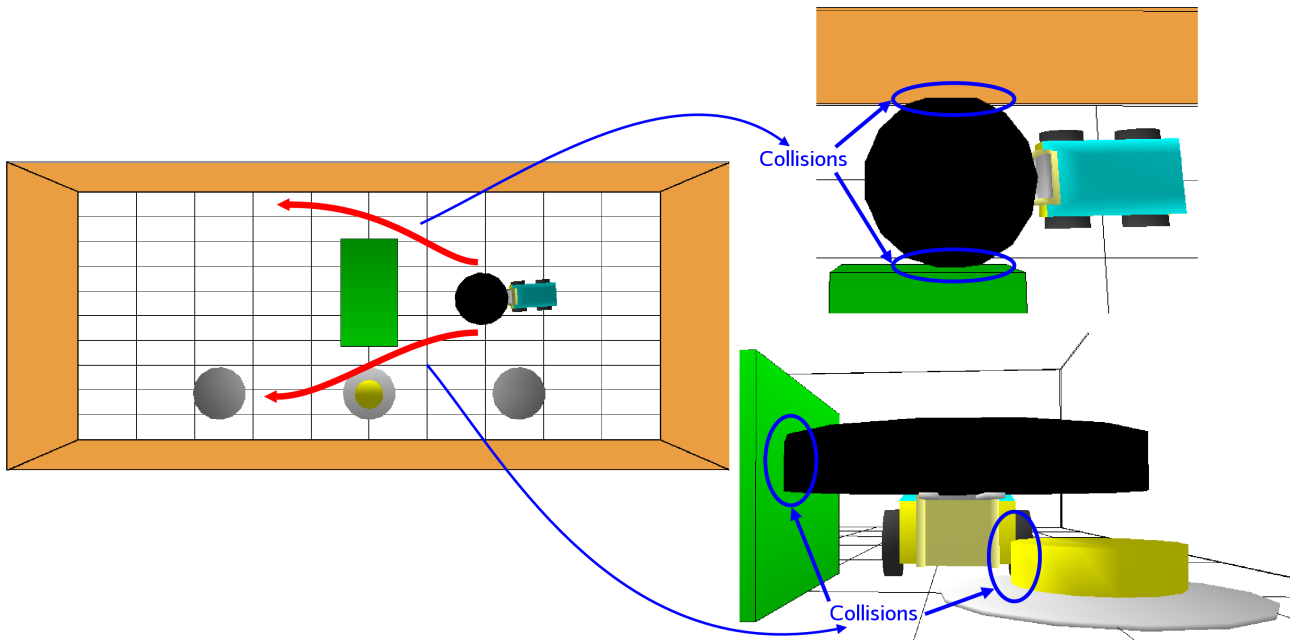


Figure 7.10: Problème de collisions pour passer le gros disque d'un coté à l'autre de la pièce quand le disque moyen est posé au centre.

robot non holonome de faire demi-tour tout en portant un objet.

Pour tous les problèmes, hormis $\mathcal{P}_3$ et $\mathcal{P}_4$, il est possible de trouver comme plan solution le plan de la figure 7.7. Ce plan fait 42 actions (il faut 6 actions pour prendre et déplacer un objet).

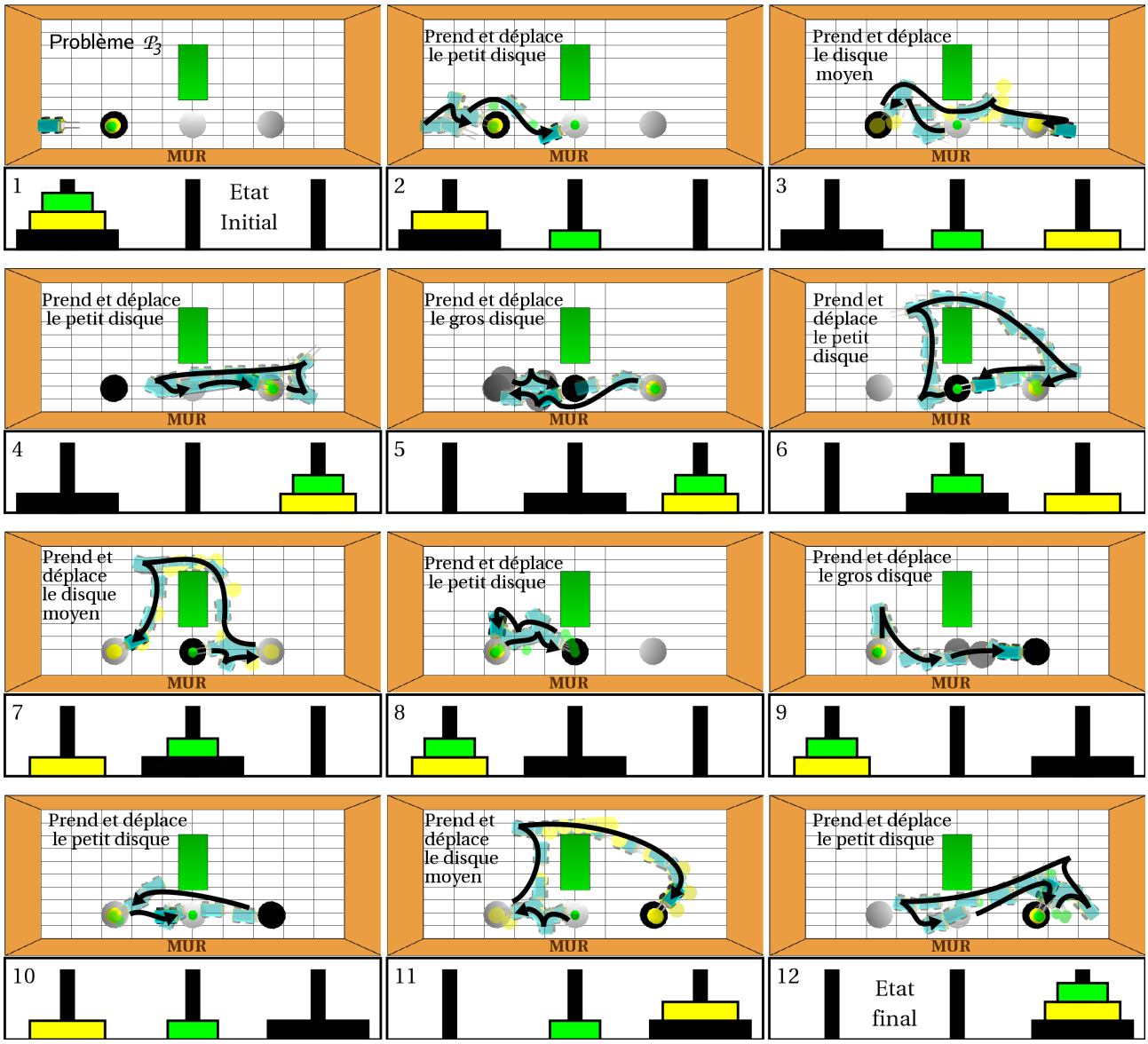


Figure 7.11: Plan solution minimal (en nombre d'actions) pour les problèmes $\mathcal{P}_3$ et $\mathcal{P}_4$.

Pour les problèmes $\mathcal{P}_3$ et $\mathcal{P}_4$ il n'est pas possible d'exécuter l'action de déplacement du gros disque en présence du disque moyen (figure 7.7 étape 4 et figure 7.10). Pour ces deux problèmes le plan solution minimal (en nombre d'actions) est montré par la figure 7.11. Ce plan fait 66 actions élémentaires.

Les résultats de la table 7.1 permettent de comparer les performances d'aSyMov sur les différents problèmes que nous avons présentés. Sur cette table est présent le taux de réussite. ASyMov est arrêté quand il a fait 1000 opérations d'extensions ou qu'il a mis plus de trois heures.

Problème:	taux de réussite	temps moyen	opérations d'extension	longueur du plan	contextes explorés
$\mathcal{P}_0$ + roadmaps	100 %	0.266 s	57.8	42	17.5
$\mathcal{P}_0$	100 %	9.97 s	95.6	43.5	25.4
$\mathcal{P}_0\,bis$	78 %	78.2 s	384.4	25.1	74.4
$\mathcal{P}_1$	98 %	135.6 s	294.6	52.6	59.3
$\mathcal{P}_2$	100 %	298.9 s	320.5	60.2	62.8
$\mathcal{P}_3$	100 %	271.6 s	365.7	66	71.2
$\mathcal{P}_4$	80 %	1850.54 s	367.8	66	69.4

Table 7.1: Résultats obtenus sur les tours de Hanoi sur la base de 50 tests (hormis $\mathcal{P}_4$ avec 5 expériences). Les moyennes sont calculées sur les planifications ayant trouvé une solution (nombre d'opération d'extension < 1000). « $\mathcal{P}_0$ + roadmaps » indique que le problème $\mathcal{P}_0$ a été résolu avec des roadmaps précalculées.

Une opération d'extension consiste à essayer de valider une action pour un état sélectionné (§ 6.2.3). Parfois seules les actions d'apprentissages sont valides, ainsi $\mathcal{P}_0$ + roadmap fait moins d'opérations d'extension que les autres problèmes. Le nombre d'opérations d'extension reflète à la fois la taille de l'espace d'état exploré par aSyMov et la difficulté à valider géométriquement les actions.

Le temps moyen permet de comparer la difficulté de résolution des problèmes. Il comprend le temps d'apprentissage de la topologie. La longueur du plan permet de voir si aSyMov trouve le plan de longueur minimale en nombre d'actions.

Le nombre de contextes explorés permet de voir les contextes pris en compte lors de la validation des actions. Ces contextes sont basés sur les positions symboliques des autres objets ou robots pris en compte pendant un déplacement.

Pour le problème $\mathcal{P}_0$, le robot se déplaçant en *Transit* pour saisir le petit disque au niveau du sol sur la seconde colonne il peut y avoir 4 contextes différents en fonction des positions du moyen et du gros disques (figure 7.11 état 2, 3, 10 et 11). Pour le problème avec un seul robot et trois objets il y a 151 mouvements possibles dans des contextes différents [1]. Ils servent à déterminer les actions possibles géométriquement.

[1] Si nous comptons tous les mouvements possibles, même ceux qui ne sont pas faisables ne partant de l'état initial nous avons la formule :

$nb-contextes = \sum^{R_i} nb-positions(R_i) \times (nb-positions(R_i)-1) \times \sum^{\mathcal{EO}_k \in \mathcal{ER}_i} \prod_{O_k \in \mathcal{EO}_j} (nb-positions(O_k)$

Avec R_i robot composé ou élémentaire, $\mathcal{ER}_i$ l'ensemble des partitions $\mathcal{EO}_j$ de robots et d'objets possibles de faire en prenant toutes les chaînes cinématiques élémentaires hormis celles de R_i. $nb-positions(X)$ représente l'ensemble des positions symboliques possibles pour le robot ou l'objet X. Cela donne respectivement 4253796 et plus d'un milliard de mouvements dans des contextes différents pour un et deux robots. Pour ne retenir que les mouvements possibles, il faut les énumérer *« à la main »* ce qui nous a donné les résultats 151 et 16148 respectivement pour un et deux robots.

Plusieurs observations peuvent être extraites de la table 7.1 :

Observation 1: Faire un apprentissage en ligne permet de limiter le nombre de contexte à tester.

Pour les problèmes avec un seul robot, le nombre de contextes possibles est de 151. Avec deux robots, ce nombre monte à 16148. Le problème $\mathcal{P}_0 bis$ cherche une solution dans un problème beaucoup plus complexe, nous avons une réduction des contextes testés encore plus flagrante.

Observation 2: Réutiliser des roadmaps déjà apprises réduit considérablement la complexité du problème.

Dans le cas $\mathcal{P}_0$ + roadmaps, nous n'avons pas d'actions d'apprentissage et aSyMov devient réellement utilisable sur des robots. Seul les opérations de validation en fonction du contexte sont coûteuses.

Observation 3: La longueur des plans trouvés dépend de la complexité géométrique.

Les problèmes $\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$ ont tous un plan de longueur minimale en 42 actions, mais plus l'environnement est contraint, plus il est difficile de trouver ce plan minimal. C'est pour cela que la moyenne de longueur des plans montre une progression pour passer de 42 actions au plan à 66 actions.

Pour les problèmes $\mathcal{P}_1$ et $\mathcal{P}_2$ aSyMov trouvent alternativement ces deux types de plan. La longueur moyenne du plan représente donc une pondération entre les deux plans, qui mesure la difficulté qu'imposent les contraintes géométriques.

Observation 4: La géométrie peut devenir très coûteuse en temps de calcul.

Les résultats de la table 7.1 montrent qu'aSyMov est capable de s'adapter à l'environnement géométrique mais quand celui-ci devient trop contraint, il peut éprouver de grandes difficultés à trouver un chemin.

Une analyse plus fine des résultats a montré que la taille des roadmaps augmentait trop pendant la planification et cela surtout pour chercher à trouver un chemin faisable pour transporter le gros objet directement de sa position initiale à sa position finale. Cela correspond à l'action la plus avantageuse du point de vue heuristique.

La difficulté dans les tours de Hanoi est que ni l'heuristique symbolique ni l'heuristique géométrique ne permettent de guider efficacement la recherche. En effet les GCEs ne font pas apparaître les collisions avec l'objet moyen.

Nous avons alors pensé à trois pistes pour de futures améliorations : un processus de validation probabiliste, des actions d'oubli, l'utilisation de *mutex*.

Dans l'implémentation actuelle d'aSyMov, le processus de validation est complet sur la topologie apprise au moment de cette validation. Mais la complexité de cet algorithme est une fonction polynomiale de la taille des listes d'accessibilité des positions géométriques (le degré du polynôme étant égal au nombre de robots et d'objets déplaçables de l'environnement). Or ces listes augmentent au cours des phases d'apprentissage. Pour un problème complexe qui nécessite de grandes phases d'apprentissage

($\mathcal{P}_4$) l'algorithme de validation devient extrêmement coûteux. De plus nous avons remarqué que cet algorithme est particulièrement lent dans le cas où il n'existe pas de solution.

Nous pensons donc qu'il serait profitable d'avoir un processus de validation qui au lieu de tester toutes les combinaisons possibles en essaie un nombre fixé. Celles-ci pourront être choisies aléatoirement, mais pour augmenter les performances, il pourrait être bon de pénaliser les combinaisons qui sont proches de choix ayant déjà échoué précédemment.

Une autre piste à suivre peut être les « actions d'oubli ». Elle sera alors le pendant des actions d'apprentissage et retirerait des roadmaps les nœuds et les arcs inutiles (n'ayant pas permis de trouver des états géométriques). Ceci permettrait de détruire une connexité non valide dans un contexte donné et donc d'utiliser l'heuristique géométrique pour guider la recherche. De plus cela diminuerait les listes d'accessibilités pour accélérer le processus de validation. Le grand problème de cette méthode est qu'il faut être sûr que les nœuds oubliés sont vraiment inutiles.

La dernière piste consisterait à s'inspirer des travaux sur « *GraphPlan* » (§ 5.2.2) qui ajoute des *mutex* afin de prendre en compte une partie des interactions qui existent entre les faits de son graphe d'accessibilité. Il pourrait être intéressant d'ajouter dynamiquement des mutex aux GCEs pour prendre en compte une partie des interactions entre les robots et les objets. Ainsi certaines composantes connexes de GCE ne pourront pas être combinées pour former de nouveaux nœuds. Ces mutex pourront être remis en cause en cas de d'enrichissement de ces composantes connexes.

7.3 Le domaine « IKEA ».

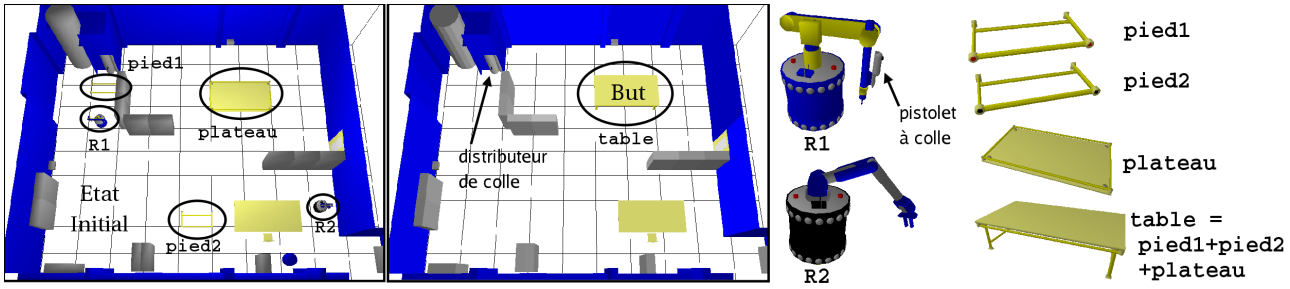


Figure 7.12: Le problème Ikea : deux robots doivent assembler une table et la poser en position but. La table est composée d'un plateau et de deux pieds. Seul le robot R1 possède un pistolet à colle pour coller les pieds au plateau. Il peut le recharger au niveau du distributeur.

Le domaine « *IKEA* » (figure 7.12) est un peu plus complexe que les autres domaines puisqu'il fait apparaître une grande variété d'actions. Il s'agit ici d'assembler un plateau avec deux pieds pour former une table et de la mettre en position finale. Les deux robots peuvent transporter les pieds de la table (figure 7.13.c). Par contre pour transporter la table assemblée il faut qu'ils se mettent à deux

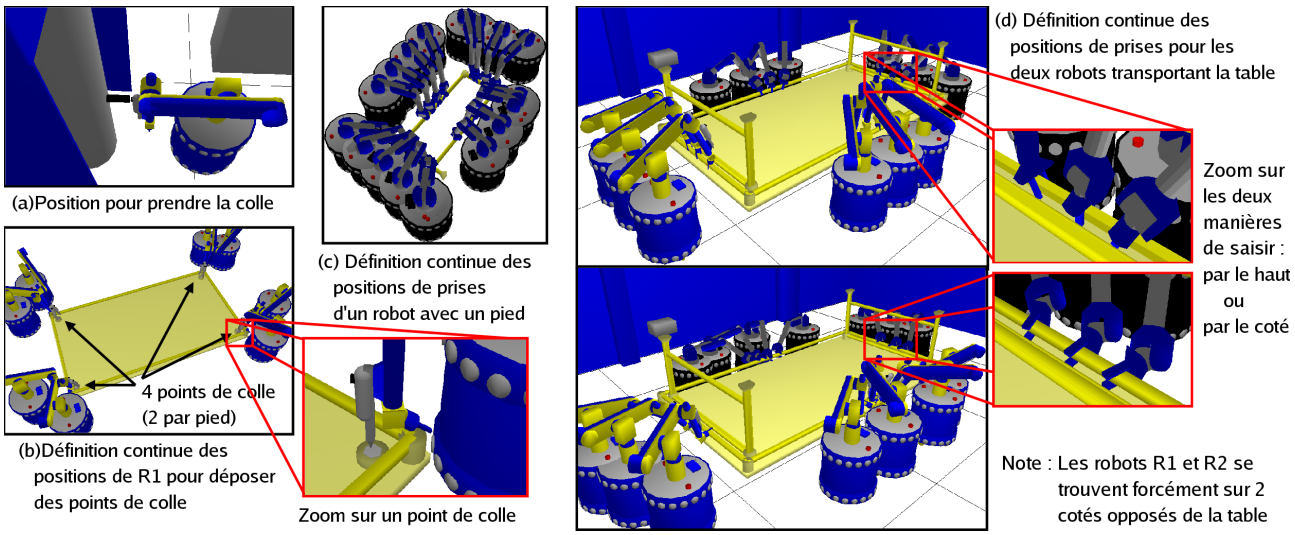


Figure 7.13: Définition géométrique des positions pour se connecter au distributeur de colle (a), pour poser des points de colle (b) et pour prendre les objets (c) et (d).

pour la porter (figure 7.13.d). De plus l'un d'entre eux (`R1`) possède un réservoir de colle. Sa réserve maximale de colle lui permet d'appliquer deux points de colle, sachant qu'il en faut au moins deux pour coller un pied (figure 7.13.b). Il peut aussi recharger sa réserve au niveau du distributeur de colle (figure 7.12 et 7.13.a).

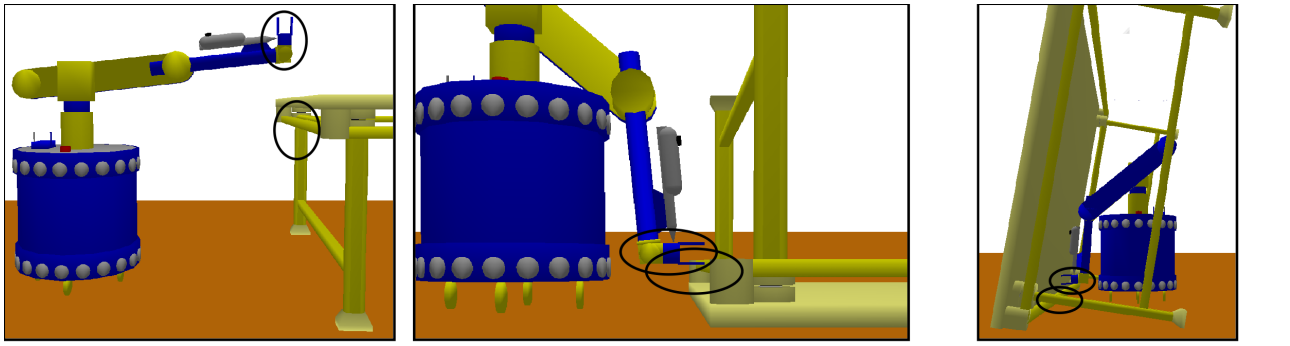


Figure 7.14: Définition des positions de prises qui ne sont pas accessibles quand la table est posée sur ses pieds (a), à l'envers (b) ou sur le coté.

Dans ce problème nous avons aussi ajouté quelques contraintes géométriques. Premièrement, la position initiale du premier pied de table ne permet pas au robot `R1` d'aller recharger son réservoir de colle. Il faut d'abord déplacer ce pied de table.

Deuxièmement les positions de prises de la table, sont telles qu'il est impossible de retourner la table en une seule opération, il faut changer de prise (figure 7.14).

Sur le plan solution (figure 7.15) nous pouvons constater que le robot `R2` met le pied de table en

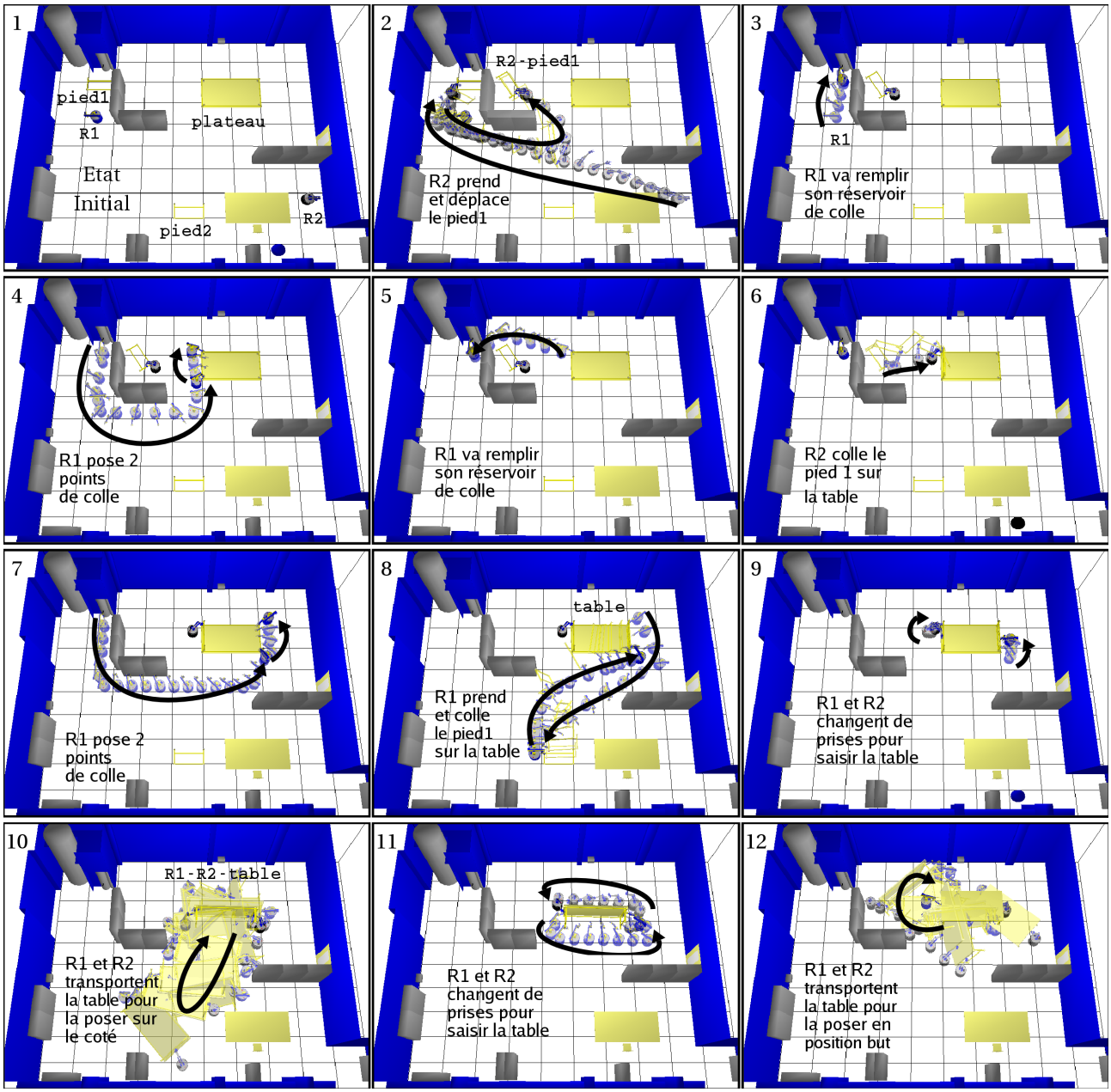


Figure 7.15: Plan solution trouvé par aSyMov sur le problème Ikea.

position intermédiaire (action 1) pour permettre au robot `R1` d'aller remplir son réservoir de colle.

Dans ce cas nous voyons une limitation de l'implémentation actuelle d'aSyMov, les positions intermédiaires des robots seuls sont des positions spécifiques. Dans cet exemple, le robot `R2-pied1` va en position pour poser le pied de table. Il ne va pas simplement s'écarter en restant en *Transfert*.

Vue la taille des roadmaps de *Transit* ou de *Transfert* des robots, autoriser une position intermédiaire qui peut être à un endroit quelconque de ces roadmaps créerait des listes d'accessibilités très importantes. Le processus de validation serait alors trop coûteux.

Comme nous l'avons remarqué précédemment, il serait possible dans des travaux futurs d'avoir un système de validation probabiliste afin de limiter les nombreux tests à chaque étape.

Une autre possibilité pour réduire cette complexité, serait de modifier le système de contraintes propagées par le processus de validation pour s'inspirer des travaux de Chadzelek[Chadzelek 96].

Il fait de la planification de mouvements pour plusieurs robots, où il cherche une solution indépendante pour chaque robot. Il détermine ensuite l'ensemble des robots $R_{obstacle}$ qui empêchent l'exécution des mouvements d'autres robots. Pour chaque robot $R_{obstacle}$ il va chercher une position intermédiaire pour que $R_{obstacle}$ à cette position n'entre pas en collision avec les trajectoires des autres robots. Il définit alors des trajectoires de dégagement. Il doit dans une dernière étape coordonner les trajectoires entre elles.

Le but d'une position intermédiaire est en fait de libérer le passage pour d'autres mouvements. Il pourrait être intéressant d'avoir une contrainte qui impose au robot ou à l'objet contraint de ne pas obstruer une partie d'une roadmaps afin de s'assurer de la faisabilité du chemin.

Plus généralement, plus les contraintes générées par le processus de validation seront fortes, plus la combinatoire des états géométriques sera faible. Par contre des contraintes fortes vont invalider des états géométriques qui pourrait être utiles. Il serait alors possible de perdre la complétude probabiliste de notre planificateur.

Chapitre 8

Conclusion.

Nous avons présenté tout au long de cette thèse, l'architecture d'aSyMov ainsi que l'ensemble des algorithmes utilisés pour la recherche d'un plan solution.

Cette recherche de solutions se fait tant par l'apprentissage de la topologie du problème que par l'exploration de l'espace des états atteignables. ASyMov a cette particularité de faire interagir fortement les techniques de planification de tâche et les techniques de planification de mouvement. Ainsi les aspects symboliques et géométriques se retrouvent à tous les niveaux du planificateur. En effet :

- ASyMov gère de manière similaire tout une gradation d'actions ayant des conséquences plus ou moins géométriques en partant des actions purement symboliques pour finir avec des actions d'apprentissage de la topologie qui n'ont aucune influence sur l'état symbolique (§ 6.4.2).
- Une action est retenue si elle est applicable (i.e. valide au sens de la planification de tâche) et qu'elle passe l'étape de validation (i.e. valide au sens de la planification de mouvement) (§ 6.3).
- Le calcul de l'heuristique prend en compte à la fois des plans trouvés par le planificateur de tâche et la connexité des roadmaps (§ 6.5).
- De ce fait le choix des états à étendre prend en compte aussi bien des paramètres géométriques que symboliques.
- L'extension des roadmaps est guidée par des règles heuristiques (§ 4.4) ayant des préconditions portant aussi bien sur des propriétés géométriques de l'environnement que sur les plans heuristiques trouvés par le planificateur de tâches.
- Le prétraitement comprend aussi bien l'analyse du domaine symbolique que l'analyse de la structuration des Graphes des Cinématiques Élémentaires ainsi que les liens qui existent entre eux.

- Le post-traitement applique des opérations dans l'espace d'état ou l'espace des plans partiels au niveau symbolique comme au niveau géométrique.

De ce fait aSyMov ne peut pas être réduit à un simple planificateur hiérarchique. L'interaction entre les techniques géométriques et symboliques étant beaucoup plus forte qu'une hiérarchie unidirectionnelle. En ce sens, il représente une approche entièrement nouvelle des techniques permettant de mettre en synergie la planification géométrique avec la planification de tâche.

Les problèmes résolus par aSyMov ne font pas intervenir un grand nombre de robots ou d'objets, mais nous pensons qu'ils correspondent au besoin d'une robotique qui veut interagir efficacement avec son environnement. Ainsi il est possible de planifier des missions qui prennent explicitement en compte les contraintes géométriques de l'environnement. Les missions du type : aller chercher un objet, monter un meuble, etc, pourront avoir des tâches élémentaires différentes, s'il y a des objets gênant ou des passages étroits.

Bien qu'apparemment simple, la complexité des problèmes résolus est très importante. Pour y parvenir aSyMov va planifier en affinant petit à petit son modèle pour finalement prendre en compte toute la complexité de celui-ci.

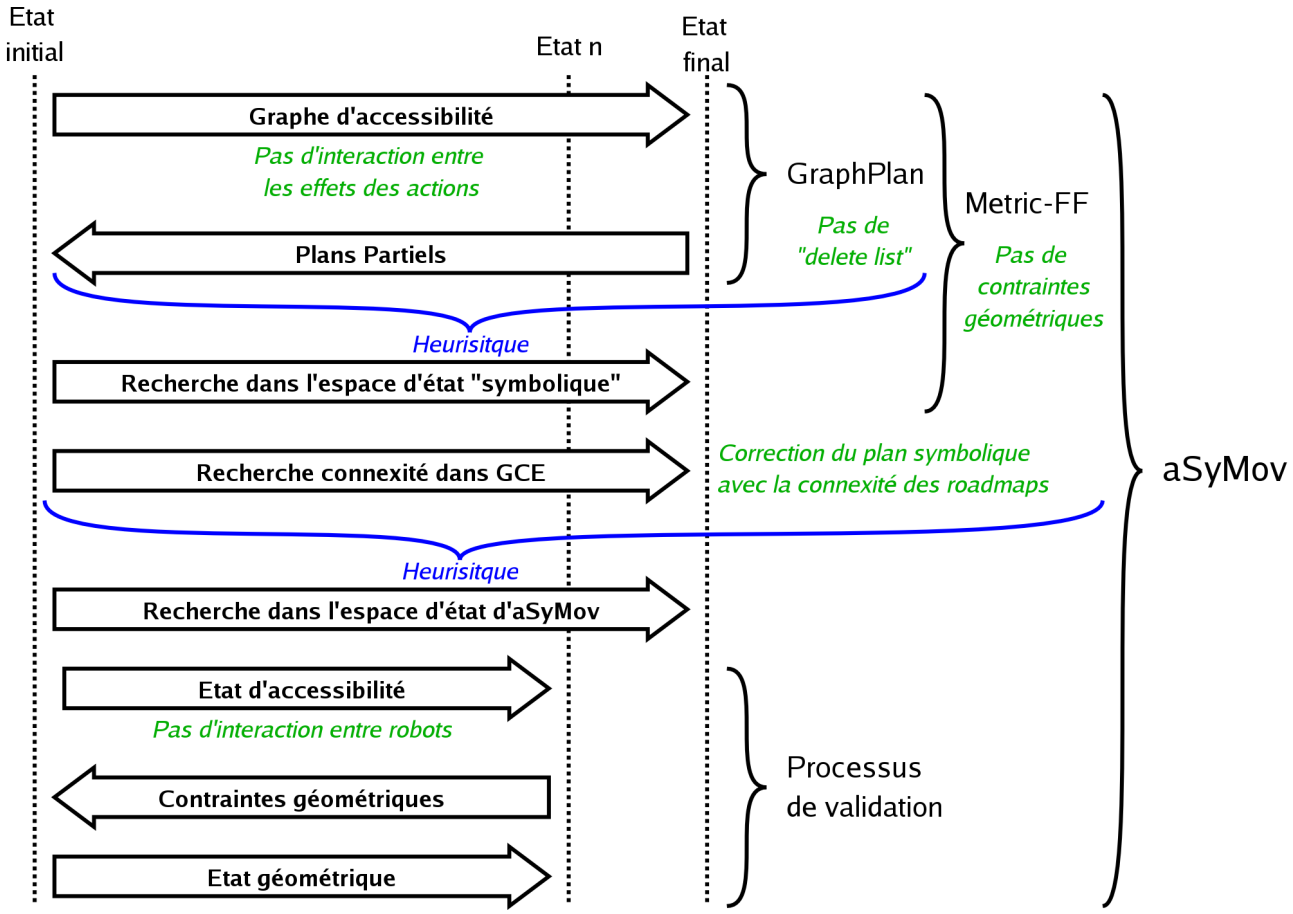


Figure 8.1: Les différentes niveaux de recherche d'aSyMov ainsi que les relaxations apportées à chaque niveau.

.

La figure 8.1 résume comment aSyMov va faire des recherches dans un problème relaxé puis ajouter progressivement toute la complexité du problème géométrique.

ASyMov est un planificateur encore très jeune. Le « gap » à franchir pour relier les notions symboliques et géométriques est très grand. Cela demande une maîtrise des techniques les plus modernes de ces deux communautés et une recherche d'algorithmes innovants pour pouvoir combiner ces deux domaines.

Plusieurs travaux futurs ont été envisagés tout au long de cette thèse. Nous pouvons rappeler ici les différents point soulevés :

- La représentation de l'état d'aSyMov. Nous avons regroupé dans un état d'aSyMov

  $$\mathcal{E} = (\mathcal{E}_S, \mathcal{E}_A, \{\mathcal{E}_{G_0}, \ldots, \mathcal{E}_{G_n}\})$$

  l'ensemble des états géométriques $\mathcal{E}_{G_i}$ atteignables suite aux actions qui ont mené à cet état $\mathcal{E}$. Il en résulte que deux états d'aSyMov $\mathcal{E}_i$ et $\mathcal{E}_j$ peuvent être différents même si leurs états symbolique $\mathcal{E}_S$ et d'accessibilité $\mathcal{E}_A$ sont égaux. Dans la version actuelle de notre système, nous avons choisi de les considérer tous différents.
  En fait les états $\mathcal{E}_i$ et $\mathcal{E}_j$ sont différents si et seulement s'il est possible de trouver un état géométrique accessible pour l'un, mais inaccessible pour l'autre.
  Il serait donc possible de diminuer la taille de l'espace de recherche en regroupant les états d'aSyMov qui n'apportent aucune différence au niveau géométrique au lieu d'augmenter le coût de recherche des états qui risquent d'être similaires (§ 6.2.2).
- La représentation des actions. Les actions d'aSyMov sont élémentaires. Il peut être intéressant de les regrouper pour former des méthodes et ainsi guider l'algorithme par des techniques *« Hierarchical Task Network »* (HTN) [Erol 95] (§ 7.1).
- L'heuristique d'aSyMov. Elle pourrait gagner en efficacité si les aspects symboliques étaient complètement intégrés. La géométrie n'est pour l'instant qu'une étape de correction.
  Un autre aspect qui peut être intéressant serait l'ajout d'informations issues de la validation au sein des GCEs. Il pourrait être intéressant d'avoir des *« mutex »* pour noter les mouvements coordonnés qui ont échoué au sein de la validation, et ainsi reporter dans le calcul de l'heuristique une partie des problèmes liés aux collisions entre robots.
- Le processus de validation. Ce processus peut être aussi amélioré par l'utilisation de nouvelles contraintes à propager afin de réduire le champ des possibilités.
  Une autre amélioration possible est de ne plus faire une validation qui teste toutes les possibilités, mais seulement une partie choisie aléatoirement. Cela permettrait d'avoir un temps de recherche un peu plus indépendant du nombre de nœuds des roadmaps.
- L'apprentissage de la topologie. Plus l'apprentissage est important, plus la taille des roadmaps est importante et plus il est difficile de trouver des chemins coordonnés. Par contre dans ces roadmaps, tous les nœuds ne sont pas égaux. Un certain nombre d'entre eux ont été ajoutés pour explorer des parties de l'espace qui n'ont aucun intérêt pour le problème à résoudre. Il peut

donc être intéressant *« d'oublier »* les nœuds inutiles pour réduire l'espace de recherche.

ASyMov est voué à évoluer et à s'améliorer notamment grâce aux travaux de la thèse de Stéphane Cambon qui font suite à ceux présentés dans ce mémoire.

Pour preuve, à l'heure où j'écris ces quelques lignes, certaines des pistes que nous avons proposées pour de futures améliorations, ont déjà été testées.

Lexique

Action applicable (§ 6.2.2) : Action qui est valide pour la partie symbolique d'un état d'aSyMov.

Action géométrique (§ 6.2.3) : Action qui possède dans ses effets des faits avec le prédicat « `on` ».

Action d'apprentissage (§ 6.4.1) : Action qui a pour but d'enrichir les roadmaps. Elle n'apparaît pas dans le plan.

Action valide (§ 6.2.3) : Action applicable qui a permis de construire un nouvel état valide d'aSyMov, c'est à dire qui a permis de construire un nouvel état géométrique.

Arc d'équivalence (§ 3.4.4) : Arc d'un GCE reliant deux nœuds qui possèdent la même composition de composantes connexes de GCE $\mathcal{G}_i$.

Arc entre roadmaps (§ 3.4.3) : Arc d'un GCE reliant deux nœuds à travers des liens d'héritage entre roadmaps.

Chaîne cinématique (§ 3.1) : Ensemble d'articulations reliées entre elles auxquelles peut être rattaché un ensemble de corps rigides.

Configuration (§ 2.1) : Instance des degrés de libertés des actionneurs d'une chaîne cinématique.

Etat d'aSyMov (§ 6.1) : Etat $\mathcal{E} = (\mathcal{E}_S, \mathcal{E}_A, \{\mathcal{E}_{G_1}, \ldots, \mathcal{E}_{G_n}\})$ regroupant l'état symbolique $\mathcal{E}_S$, l'état d'accessibilité $\mathcal{E}_A$ et un ensemble d'états géométriques $\mathcal{E}_{G_i}$ possibles.

Etat symbolique (§ 6.1.1) : Partie d'un état d'aSyMov utilisée par le planificateur symbolique.

Etat d'accessibilité (§ 6.1.2) : Etat représentant l'ensemble des instances géométriques possibles des positions symboliques en fonction de la connexité des roadmaps et des actions effectuées. Il est représenté par l'ensemble des nœuds accessibles des positions symboliques appliquées.

Etat géométrique (§ 6.1.3) : Instance géométrique d'un état symbolique. Il est représenté par un ensemble de nœuds correspondants aux positions symboliques de l'état d'aSyMov tel qu'il existe un ensemble de mouvements valides à travers les roadmaps qui correspondent aux actions précédentes de l'état et qui mène à cet état géométrique.

Graphe des Cinématiques Elémentaires (GCE) (§ 3.4) : Graphe permettant de relier entre elles les composantes connexes des roadmaps de mouvement et des roadmaps affinables pour un ensemble de chaînes cinématiques élémentaires.

Grasp $\cap$ ***Placement*** (§ 2.7.2) : Espace de recherche où l'objet est saisi par un robot et est aussi en position stable. Dans cet espace il est possible de faire des variations continues des positions de prise. Un chemin trouvé dans *Grasp* $\cap$ *Placement* pourra être transformé en une séquence fini de mouvements de *Transit* et de *Transfert*.

Lien d'héritage (§ 3.3) : Liens entre les nœuds de différentes roadmaps permettant de reporter tout ou partie de la configuration d'un noeud source dans une roadmap de destination. Cela permet de faire des changements de roadmap (méthode locale) pour une même configuration de robot ou d'objet. Par exemple un noeud de *Transfert* peut être lié à un noeud de *Transit* au moment où le robot lâche un objet.

Méthode locale (§ 2.2.1) : Une *« méthode locale »* permet de tester l'existence d'un chemin faisable entre deux points donnés d'une roadmap. Elle définit le mouvement d'un robot entre ces deux configurations en prenant en compte les différentes contraintes mécaniques et cinématiques du système.

Nœud de composition (§ 3.4.2.2) : Nœud d'un GCE $\mathcal{G}$ représentant des composantes connexes de GCE $\mathcal{G}_i$ formant une partition de $\mathcal{G}$.

Nœud de roadmap (§ 3.4.2.1) : Nœud d'un GCE représentant une composante connexe d'une roadmap de mouvement ou d'une roadmap affinable.

Objet déplaçable (§ 3.1.1) : Chaîne cinématique avec des corps rigides, ne possédant aucune méthode de mouvement propre.

Position symbolique (§ 5.3.4) : Elle associe une fonctionnalité à un sous-ensemble de l'espace des configurations. Elle est représentée par l'ensemble des nœuds des roadmaps satisfaisant les critères de cette fonction (nœud pour saisir un objet, ouvrir une porte, se déplacer, ...).

Position symbolique appliquée (§ 5.3.4) : Position symbolique dont l'ensemble des nœuds est restreint au sous-ensemble des nœuds atteignables par connexité en fonction de l'historique des actions précédentes.

Roadmap (§ 2.2.1) : Une *« roadmap »* est un graphe orienté ou non suivant la nature de la *« méthode locale »*, dont les nœuds correspondent à des configurations choisies aléatoirement et les arcs correspondent aux *« méthodes locales »*.

Roadmaps affinables (§ 3.2.2) : Elles représentent une propriété de faisabilité pour un ensemble de mouvements. Les mouvements réels étant calculés en fin de traitement.

Roadmaps heuristiques (§ 4.1) : Elles ne représentent pas de mouvements faisables mais sont utilisées pour aider la construction d'autres roadmaps.

Roadmaps de mouvements (§ 3.2.1) : Elles représentent les mouvements faisables par les robots, ou par les objets.

Roadmaps relatives (§ 4.3) : Elles représentent les mouvements faisables par les robots, ou par les objets.

Robot (§ 3.1.1) : Chaîne cinématique avec des corps rigides, possédant des méthodes locales représentant des mouvements autonomes.

Références bibliographiques

[Ahuactzin 92] J.M. Ahuactzin, E.-G. Talbi, P. Bessière & E. Mazer. *Using Genetic Algorithms for Robot Motion Planning.* In 10*th* Europ. Conf. Artific. Intelligence, pages 671–675, London, England, 1992.

[Ahuactzin 94] J.-M. Ahuactzin. *Le Fil d'Araine: Une Méthode de Planification Générale. Application à la Planification Automatique de Trajectoires.* PhD thesis, L'Institut National Polythechnique de Grenoble., Septembre 1994.

[Ahuactzin 95] J.M. Ahuactzin & K. Gupta. *Using Manhattan Paths for Manipulation Planning.* Third IASTED International Conference on Robotics and Manufacturing, Cancùn, México, 1995.

[Ahuactzin 98] J.-M. Ahuactzin, K. Gupta & E. Mazer. *Manipulation Task Planning for Redundant Robots: A Practical Approach.* In K. Gubta & A.P. Del Pobil, editeurs, Practical Motion Planning in Robotics: Current Approches and Future Directions, pages 79–113. J. Wiley, 1998.

[Alami 89] R. Alami, T. Siméon & J-P. Laumond. *A Geometrical Approach to Planning Manipulation Tasks. The case of discrete placements and grasps.* In Robotics Research: The Fifth International Symposium,Tokyo, August 1989.

[Alami 94] R. Alami, J.P. Laumond & T. Siméon. *Two Manipulation Planning Algorithms.* The first Workshop on the Algorithmic Foundations of robotics.(WAFR'94),San Fransisco, pages 109–125, February 1994.

[Bacchus 96] F. Bacchus & F. Kabanza. *Using Temporal Logic to Control Search in a Forward Chaining Planner.* In IOS Press, editeur, New Directions in Planning, pages 141–153. M. Ghallab and A. Milani, 1996.

[Bessière 93] P. Bessière, J.M. Ahuactzin, E.-G. Talbi & E. Mazer. *The "Ariadne's Clew" Algorithm: Global Planning with Local Methods.* IEEE Internationnal Conference on Intelligent Robots and Systems (IROS), vol. 2, pages 1373–1380,

1993.

[Blum 97] A.L. Blum & M.L. Furst. *Fast planning through planning graph analysis.* Artificial Intelligence, pages 281–300, 1997.

[Bonet 01] B. Bonet & H. Geffner. *Planning as Heuristic Search.* Artificial Intelligence, Special issue on Heuristic Search, vol. 129, 2001.

[Boor 99] V. Boor, M.H. Overmars & A.F. van der Stappen. *The Gaussian Sampling Strategy for Probabibilistic Roadmap Planners.* Proc. IEEE International Conference on Robotics and Automation, pages 1018–1023, 1999.

[Brock 97] O. Brock & O. Khatib. *Elastic Strips: Real-Time Path Modification for Mobile Manipulation.* The English International Sympotium of Robotics Research, 1997.

[Canny 87] J.F. Canny. *The Complexity of Robot Motion Planning.* PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Mai 1987.

[Chadzelek 96] Thomas Chadzelek. *Heuristic Motion Planning with Many Degrees of Freedom.* 8th Canadian Conference on Computational Geometry, 8 1996.

[Chapman 87] D. Chapman. *Planning for conjunctive goals.* Artificial Intelligence, no. 32, pages 333–377, 1987.

[Cortés 02] J. Cortés, T. Siméon & J.P. Laumond. *A Random Loop Generator for Planning the Motions of Closed Kinematic Chains using PRM Methods.* IEEE International Conference on Robotics and Automation, May 2002.

[Cortés 03] J. Cortés & T. Siméon. *Probabilistic Motion Planning for Parallel Mechanisms.* IEEE International Conference on Robotics & Automation, 2003.

[Doherty 99] P. Doherty & J. Kvarnstrom. *Talplanner : An empirical investigation of temporal logic-based forward chaining planner.* International Workshop on Temporal Representation and Reasoning, 1999.

[D.S. Nau 99] A. Lotem D.S. Nau Y. Cao & H. Muñoz-Avila. *Shop: Simple hierarchical ordered planner.* International Joint Conference on Artificial Intelligent, 1999.

[Erol 95] K. Erol. *HTN Planning: Formalization, Analysis, and Implementation.* PhD

thesis, University of Maryland, 1995.

[F. Lamiraux 02] C. Van Geem F. Lamiraux D. Bonnafous. *Path Optimization for Nonholonomic Systems: Application to Reactive Obstacle Avoidance and Path Planning.* In A. Bicchi, H. Christensen & D. Prattichizzo, editeurs, Control Problems in Robotics. 2002.

[Fade 90] B. Fade & P. Regnier. *Temporal Optimization of Linear Plans of Actions: A Strategy Based on a Complete Method for the Determination of Parallelism.* Rapport technique, 1990.

[Ferbach 94] P. Ferbach & J. Barraquand. *Path Planning through Variational Dynamic Programming.* Rapport technique 33, Paris Research Laboratory, Digital Equipement Coorporation, 1994.

[Ferbach 96] P. Ferbach. *Contribution à la Planification de Trajectoires.* PhD thesis, Ecole Polytechnique en collaboration avec Electricité De France (EDF), France, 1996.

[Fikes 71] R.E. Fikes & N.L. Nilson. *Learning and executing generalized robot plans.* Artificial Intelligence, pages 251–288, 1971.

[Fox 99] M. Fox & D. Long. *The detection and exploration of symmetry in planning problems.* International Joint Conference on Artificial Intelligent, 1999.

[Fox 01] M. Fox & D. Long. *PDDL2.1: An extension to PDDL for expressing temporal planning domains.* Rapport technique, Univ. of Durham, UK, 2001.

[Ghallab 89] M. Ghallab & A. Mounir-Alaoui. *Managing efficiently temporal relations through indexed spanning trees.* International Joint Conference on Artificial Intelligent, 1989.

[Gravot 01] F. Gravot & R. Alami. *An extention of the Plan-Merging Paradigm for multi-robot coordination.* IEEE International Conference on Robotics & Automation, May 2001.

[Gravot 03] F. Gravot & R. Alami. *A method for handling multiple roadmaps and its use for complex manipulation planning.* IEEE International Conference on Robotics & Automation, 2003.

[Gupta 98] K. Gupta & A.P. Del Pobil (eds). Practical motion planning in robotics: Current approches and future directions. J. Wiley and Sons, 1998.

[Han 00] L. Han & N.M. Amato. *A Kinematics-based Probabilistic Roadmap Method for Closed Chain Systems.* Workshop on Algorithmic Foundations of Robotics (WAFR'00), pages 233–246, 2000.

[Hoffmann 01] J. Hoffmann & B. Nebel. *The FF Planning System: Fast Plan Generation Through Heuristic Search.* Journal of Artificial Intelligence Research, 2001.

[Hoffmann 02] J. Hoffmann. *Extending FF to Numerical State Variables.* In Proceedings of the 15th European Conference on Artificial Intelligence, France, Lyon, July 2002.

[Hsu 98] D. Hsu, L.Kavraki, J.-C. Latombe, R. Motwani & S. Sorkin. *On Finding Narrow Passage with Probabilistic Roadmap Planner.* In Robotics: The Algorithmic Perspective (WAFR98), P. Agarwal et al. (Eds). AK Peters, 1998.

[J. Barraquand 92] B. Langlois J. Barraquand & J.C. Latombe. *Numerical potential field techniques for robot path planning.* IEEE Transactions on Systems, Man. and Cybernetics, 1992.

[J.-P. Laumond 94] M. Tax. J.-P. Laumond P.E. Jacobs. & R.M. Murray. *A motion planner for nonholonomic mobile robots.* IEEE International Conference on Robotics and Automation, pages 577–593, 1994.

[Kavraki 95] L. Kavraki. *Random Networks in Configuration Space for Fast Path Planning.* PhD thesis, Standford University, 1995.

[Kavraki 96] L. Kavraki, P. Švestka, J.-C. Latombe & M. Overmars. *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces.* IEEE Int'l Conf. on Robotics and Automation (ICRA'1996), pages 566–580, 1996.

[Khatib 86] O. Khatib. *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots.* International Journal of Robotics Research, vol. 5(1), pages 90–98, 1986.

[Koehler 98] J. Koehler. *Planning under ressource constraints.* European Conference on Artificial Intelligent, pages 489–493, 1998.

[Koga 94] Y. Koga, K. Kondo, J.J. Kuffner & J.-C. Latombe. *Planning Motions with Intentions.* Proc. SIGGRAPH'94, pages 395–408, 1994.

[Koga 95] Y. Koga. *On Computing Multi-Arm Manipulation Trajectories.* PhD thesis, Mechanical Engineering Dept, Stanford University, February 1995.

[Kuffner 00] J.J. Kuffner & S.M. LaValle. *RRT-Connect: An Efficient Approach to Single-Query Path Planning.* IEEE Int'l Conf. on Robotics and Automation (ICRA'2000),San Francisco, 2000.

[Laborie 95] P. Laborie. *IxTeT : une approche intégrée pour la gestion de ressources et la synthèse de plans.* PhD thesis, LAAS-CNRS, 1995.

[Lamiraux 01] F. Lamiraux & J.P. Laumond. *Smooth motion planning for car-like vehicles.* IEEE Transactions on Robotics and Automation, Août 2001.

[Latombe 91] J.-C. Latombe. Robot motion planning. Kluwer Academic Publishers, Boston,MA, 1991.

[Laumond 89] J.-P. Laumond & R. Alami. *A Geometric Approach to Planning Manipulation Tasks: the case of a circular robot and a movable circular object admist polygonal obstacles.* Rapport technique Rep. No. 88314, LAAS-CNRS, 1989.

[Laumond 94] J.P. Laumond, P.E. Jacobs, M. Taix & R.M. Murray. *A Motion Planner for Nonholonomic Mobile Robots.* IEEE International Conference on Robotics and Automation, vol. 10(5), pages 577–593, 1994.

[Laumond 98] J.-P. Laumond. Robot motion planning and control. Springer Verlag, 1998.

[Laumond 00] J.-P. Laumond & T. Siméon. *Notes on Visibility Roadmaps and Path Planning.* Workshop on the Algorithmic Foundations of robotics.(WAFR'00), Hannover (USA), pages 67–77, March 2000.

[LaValle. 98] Steven M. LaValle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning.* Technical Report No. 98-11, Dept. of Computer Science, Iowa State University, 1998.

[LaValle 99a] S.M. LaValle & J.J. Kuffner. *Randomized Kinodynamic Planning.* Proc. IEEE International Conference on Robotics and Automation, 1999.

[LaValle 99b] S.M. LaValle, J. Yakey & L.E. Kavraki. *A Probabilistic Roadmap Approach for Systems with Closed Kinematic Chains.* Proc. IEEE International Conference on Robotics and Automation, pages 1671–1676, 1999.

[LaValle 00] S.M. LaValle & J.J. Kuffner. *Rapidly-Exploring Random Trees: Progress and Prospects.* Workshop on the Algorithmic Foundations of Robotics, 2000.

[LaValle 01] S.M. LaValle & J.J. Kuffner. *Randomized Kinodynamic Planning.* Interna-

tional Journal of Robotics Research, vol. Vol. 20, no. No. 5, pages 378–400, 2001.

[Lozano-Pérez 83] T. Lozano-Pérez. *Spatial Planning: A Configuration Space Approach.* IEEE Transaction on computer, vol. 32(2), pages 108–120, 1983.

[M. Ghallab 98] C. Knoblock D. McDermott A. Ram M. Veloso D. Weld M. Ghallab A. Howe & D. Wilkins. *"Pddl" - the planning domain definition language.* Rapport technique, 1998.

[Mazer 98] E. Mazer, J.M. Ahuactzin & P. Bessière. *The Ariadne's Clew Algorithm.* Journal of Artificial Intelligence Research, pages 295–316, 1998.

[McAllester 91] D. McAllester & D. Rosenblitt. *Systematic non-linear planning.* American National Conference on Artificial Intelligence, 1991.

[Nielsen 97] J. Nielsen & B. Roth. *Formulation and Solution of the Direct and Inverse Kinematics Problem for Mechanism and Mechatronic Systems.* In Proceedings of the NATO Advanced Study Institute on Computational Methods in Mechanisms, vol. 1, pages 233–252, 1997. Bulgaria.

[Nilson 80] N.J. Nilson. Principles of artificial intelligence. Springer-Verlag, 1980.

[Nilsson 69] N.J. Nilsson. *A Mobile Automation: An Application of Artificial Intelligence Techniques.* Proceedings of the 1st International Joint Conference on Artificial Intelligence, pages 509–520, 1969.

[Nissoux 99a] C. Nissoux. *Visibilité et Méthodes Probabilistes pour la Planification de Mouvement en Robotique.* PhD thesis, LAAS-CNRS, Universté Paul Sabatier de Toulouse, Décembre 1999.

[Nissoux 99b] C. Nissoux, T. Siméon & J.-P. Laumond. *Visibility Based Probabilistic Roadmaps.* IEEE International Conference on Intelligent Robots and Systems, pages 1316–1321, 1999.

[O'Dùnlaing 82] C. O'Dùnlaing & C.K. Yap. *A Retraction Method for Planning The Motion of a Disc.* Journal of Algorithms, vol. 6, pages 104–111, 1982.

[O'Donnell 89] P.A. O'Donnell & T. Lozano-Perez. *Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators.* IEEE Robotics and Automation Conference, pages 484–489, May 1989.

[Oliver 00] Spence Oliver, Mahesh Saptharishi, John Dolan, Ashitey Trebi-Ollennu & Pradeep Khosla. *Multi-robot Path Planning by Predicting Structure in a Dynamic Environment.* In Proceedings of the First IFAC Conference on Mechatronic Systems, volume II, pages 593–598, September 2000.

[Qutub 98] S. W. Qutub. *Un Paradigme Générique pour la Navigation Coopérative de Robots Mobiles Autonomes.* PhD thesis, Université Paul Sabatier de Toulouse, 1998.

[R. Alami 98] F. Ingrand R. Alami & S. Qutub. *A Scheme for Coordinating Multi-robot Planning Activities and Plans Execution.* European Conference on Artificial Intelligent, 1998.

[R.E. Fikes 72] N.L. Nilson R.E. Fikes P.E. Hart. *Learning and executiong generalized robot plans.* Artificial Intelligence, 1972.

[Régnier 92] P. Régnier. *Recherche et exploitation du parallélisme en planification.* PhD thesis, IRIT, UPS Toulouse, mai 1992.

[Reif 79] J.H. Reif. *Complexity of the Mover's Problem and Generalisations.* In Proc. 20th IEEE Symposium on Foundations of Computer Science (FOCS), pages 421–427, 1979.

[S. Qutub 97] F. Ingrand S. Qutub R. Alami. *How to Solve Deadlock Situations within the Plan-Merging Paradigm for Multi-robot Cooperation.* IEEE International Conference on Intelligent Robot & System, 1997.

[Sahbani 03] Anis Sahbani. *Planification de tâches de manipulation en robotique par des approches probabilistes.* PhD thesis, Université Paul Sabatier de Toulouse, 2003.

[Schwartz 83] J.T. Schwartz & M. Sharir. *On The Piano Mover's Problem: I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Admist Polygonal Barriers.* Communications on Pure and Applied Mathematics, vol. 36, pages 345–398, 1983.

[Siméon 00] T. Siméon, J.-P. Laumond & C. Nissoux. *Visibility Based Probabilistic Roadmaps for Motion Planning.* Advanced Robotics Journal, 2000.

[Svestka 98] P. Svestka & M. Overmars. *Coordinated path planning for multiple robots.* Robotics and Autonomous Systems, 1998.

[Švestka 96] P. Švestka. *On Probabilistic Completeness and Expected Complexity of Probabilistic Path Planning.* Rapport technique, Departement of Computer Science, Utrecht University, 1996.

[Warren 90] C. W. Warren. *Multiple robot path coordination using artificial potentiel fields.* IEEE International Conference on Robotics & Automation, 1990.

[Wilfong 88] G. Wilfong. *Motion Planning in The Presence of Movable Obstacles.* Proceedings of the 4th ACM Symposium on Computational Geometry, pages 279–288, 1988.

[Wilmarth 99a] S.A. Wilmarth, N.M. Amato & P.F. Stiller. *MAPRM: A Probabilistic Roadmap planner with sampling on the Medial Axes of the Free Space.* IEEE International Conference on Robotics and Automation, pages 1024–1031, 1999.

[Wilmarth 99b] S.A. Wilmarth, N.M. Amato & P.F. Stiller. *Motion Planning for a Rigid Body Using Random Networks on the Medial Axes of the Free Space.* ACM Symposium on Computational Geometry, pages 173–180, 1999.

[Yakey 99] J. Yakey. *Randomized Path Planning for Linkages with Closed Kinematic Chains.* PhD thesis, Computer Science Departement, Iowa State University, 1999.

aSyMov : Fondation d'un planificateur robotique intégrant le symbolique et le géométrique.

Résumé :
Les travaux présentés dans cette thèse portent sur la planification de tâches pour des systèmes robotiques qui prennent en compte des contraintes géométriques, cinématiques et symboliques. Nous souhaitons notamment traiter des problèmes nécessitant la manipulation et l'assemblage d'objets par plusieurs robots mobiles manipulateurs dans un environnement contraint.

Nous avons développé un planificateur nommé « aSyMov » qui offre un cadre innovant pour combiner les techniques issues de la planification de mouvements et de la planification de tâches.

Dans un premier temps nous faisons un état de l'art des techniques de la planification de mouvements. Puis une nouvelle représentation de l'espace de recherche et de nouvelles techniques de planifications seront développées afin de généraliser et de combiner les algorithmes décrits dans l'état de l'art. Dans un second temps nous abordons les techniques de la planification de tâches et définissons une manière originale de lier le domaine géométrique au domaine symbolique. Puis nous présentons l'architecture et les algorithmes de notre planificateur.

ASyMov effectue une recherche *en avant* dans l'espace d'état. L'état d'aSyMov a la particularité de représenter l'ensemble des instanciations géométriques possibles pour un état symbolique donné. La procédure de validation des actions va essayer de minimiser les instances géométriques à vérifier. Ce ne sera que quand les actions vont contraindre la géométrie qu'une propagation arrière de ces contraintes sera faite pour trouver de nouvelles instances valides.

Nous décrivons aussi comment il est possible de combiner à la fois l'apprentissage de la topologie de l'environnement et la recherche d'une solution avec les connaissances déjà acquises.

Pour finir nous présentons plusieurs problèmes complexes qui ont été résolus par aSyMov.

Mots-clés : Planification de mouvement, planification de tâches, méthodes probabilistes, manipulation, robotique.

aSyMov : Foundation of a geometric and symbolic robotic planner.

Abstract:
In this thesis, we address the problem of planning for robotic systems by facing its full complexity : we take into account geometric, kinematic and symbolic constraints. For instance, we are interested in problems where several mobile manipulators robots have to manipulate and assemble objects in a constrained environment.

Whereas task planning and motion planning have been solved independently so far, we propose to consider these problems as a whole and to combine specific techniques in a single planner. To reach this goal, we introduce a new search space and propose original search algorithms. This work has been validated through the development of a system called "aSyMov" and we report encouraging results on complex problems.

The report consists in five main chapters. First we describe state-of-the-art motion planning methods. Then a new search space and new search techniques are introduced to generalize and combine the previous algorithms. This is followed by a summary of the most recent techniques in task planning and by the description of an original way to bind geometric and symbolic domains. Then we turn to the presentation of the architecture and the algorithms of our planner.

ASyMov is a *forward* search planner in the state space. A *state* in aSyMov actually stands for all the possible geometric instances defined by a given set of symbolic facts. At each step of the search, the chosen action is validated through a procedure carefully designed to reduce the number of geometric instances to check. This is achieved by triggering back-propagation of geometric constraints only for *geometric* actions.

A strong advantage of our solution is the possibility to interwine the learning of the environment topology (through motion planning techniques) and the search of a solution.

Finally we show several problems solved by aSyMov: these original examples necessitate to consider both geometric and symbolic aspects and thus illustrate the strength of our contribution.

Keywords: Motion planning, task planning, probabilistic methods, manipulation, robotic.