Planifier avec les contraintes géométriques du mouvement et de la manipulation

Stéphane Cambon

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 : Intelligence Artificielle

par

Stéphane Cambon

Planifier avec les contraintes géometriques du mouvement et de la manipulation

---

Raja CHATILA	**LAAS-CNRS, Toulouse**	Président
Michael BEETZ	**TUM, Munich**	Rapporteur
Christian LAUGIER	**INRIA, Saint Ismier**	Rapporteur
Rachid ALAMI	**LAAS-CNRS, Toulouse**	Directeur de thèse
Malik GHALLAB	**LAAS-CNRS, Toulouse**	Directeur de thèse
? ?	**? ?**	Examinateur
? ?	**? ?**	Examinateur

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

Table des matières

Table des figures

Liste des tableaux

Chapitre I

Problématique

I.1 Introduction

Au sein du groupe de Robotique et Intelligence Artificielle du LAAS-CNRS, nous tentons de doter les robots d'autonomie. Cela signifie qu'ils doivent être capables d'accomplir seuls des missions proposées par l'homme dans un environnement inconnu ou partiellement connu. Le robot devra donc être capable de percevoir son environnement et de raisonner pour accomplir ses missions.

Une approche proposée par notre groupe est souvent présentée sous la forme d'une architecture logicielle à trois niveaux [Alami 98]. Le premier niveau est appelé niveau décisionnel. Il comporte des composants logiciels permettant de décider des prochaines tâches que le système doit accomplir. Le second niveau est appelé niveau de supervision d'exécution. Il s'assure du bon déroulement des actions proposées par le niveau supérieur et le cas échéant en informera le niveau décisionnel. Enfin, le dernier niveau est dit "niveau fonctionnel". Il regroupe les fonctions utilisant les capacités sensori-motrices du robot (commandes des moteurs, gestion des senseurs, etc) et les sous-systèmes spécialisés dans l'accomplissement d'une tâche (reconnaissance de visages, génération de la parole, etc). Chaque fonction élémentaire constitue un module qui peut communiquer avec le niveau de supervision d'exécution pour renvoyer des bilans ou avec d'autres modules.

Cette thèse se situe au niveau décisionnel. Pour prendre des décisions, on se munit souvent d'une représentation abstraite du monde que l'on peut appeler modèle. Ce modèle représente les capacités du robot que l'on appellera les *actions* ou *tâches*, l'environnement et comment l'environnement peut être modifié par les *actions*. Pour prendre la décision de la prochaine action à mettre en oeuvre, on raisonnera sur cette représentation abstraite. Une approche classique consiste à calculer un plan d'actions garantissant le succès de la mission dans le monde abstrait. Ce calcul est réalisé par un système dit *planificateur*. Ce plan sera alors soumis au reste de l'architecture et devra être revu à chaque désaccord entre le monde abstrait et l'environnement réel dans lequel le robot évolue.

Il paraît raisonnablement impossible d'avoir un modèle parfait capable de prévoir sans failles les bonnes actions à accomplir pour réussir une mission. Dans l'idéal, des allers retour d'informations au sein de l'architecture peuvent permettre au niveau décisionnel de corriger son modèle ou de proposer de nouvelles décisions.

Plus le modèle est simple, plus le plan d'actions sera facile à calculer. Un modèle "trop" simple sera par contre incapable de prendre les "bonnes" décisions. Par exemple, une représentation du monde dans laquelle on ne modélise pas le temps sera inapte à prendre des décisions pour un robot d'exploration planétaire. En effet dans ce type d'application, le temps est une donnée fondamentale pour ne pas rater de fenêtre de communication avec un orbiter ou pour savoir de combien de lumière on pourra encore disposer pour recharger les batteries. Un robot d'exploration planétaire muni d'un modèle atemporel échouera très certainement et rapidement sa mission.

Pour un système et pour une série de missions possibles, il faut donc choisir leprésentation abstraite la plus équilibrée. Elle doit prendre en compte les aspects fondamentaux du monde réel pour une "bonne" prise de décision. Elle doit laisser de côté les autres aspects pour pouvoir prendre ces décisions plus rapidement.

Les robots sont des systèmes mobiles, c'est leur principal intérêt. La plupart des utilisations que l'on peut imaginer pour les robots sont basées sur l'utilisation de leurs mobilités. Qui plus est, dans l'idéal, ils auraient même des capacités de manipulation d'objets. Les tâches qu'un robot doit être capable de réaliser implique donc souvent des mouvements. Dans un tel cadre, nous pensons que le calcul de la faisabilité des mouvements doit être pris en compte au plus haut niveau. Dans cette thèse nous proposons un modèle et un planificateur associé répondant à cette attente.

Plus précisément, nous souhaitons définir dans cette thèse un modèle et un planificateur associé pour un ensemble de robots capable entre autre de manipuler un nombre fini d'objet. Pour prendre de "bonnes" décisions dans ce cas, le modèle devra certainement prendre en compte des aspects :

– géométriques : les mouvements et les manipulations des robots ne doivent pas provoquer de collisions dans l'environnement.

– de ressources : pour gérer l'énergie, la mémoire ou la CPU des robots.

– relationnels : pour doter le robot de raisonnement de haut niveau. Mettre en relation par exemple une clef avec une porte pour représenter le fait que la clef permet d'ouvrir la porte.

– physiques et dynamiques : pour manipuler des objets, il faut avoir des garanties sur la faisabilité physique des actions. Pour saisir un verre par exemple, il faut avoir une bonne prise, ne pas le serrer trop fort, avoir les doigts qui "accrochent"...

– temporels : pour gérer des contraintes de rendez-vous, de dates à ne pas dépasser, etc.

Raisonner sur autant d'aspects est irréalisable en l'état actuel des connaissances. Le but de cette thèse est surtout d'étudier les interactions existantes entre les aspects géométriques et relationnels d'un problème de planification. Cette interaction est souvent négligée alors qu'elle peut être cruciale.

Dans notre modèle, nous ne garderons alors que les aspects géométriques, relationnels et dans une certaine mesure des aspects de ressources. Un plan d'action calculé sur ce modèle ne garantira pas le respect des autres contraintes. Toutefois pour certains aspects physiques et dynamiques, nous pourrons prendre en compte certaines contraintes grâce à des prétraitements. Par exemple dans le cas d'un robot manipulant un verre, on peut trouver avantage à résoudre le problème de trouver une prise correcte du verre en prétraitement. La prise trouvée sera transformée en une donnée géométrique. Les prises géométriques seront des données d'entrée de notre modèle. Les forces à appliquer seront déléguées à un autre sous-système.

Les aspects temporels eux, sont simplifiés pour limiter la complexité du problème : au lieu d'un temps quantitatif explicite, on ne traitera que d'une suite asynchrone d'événements et d'états. De nombreuse études ont déjà été faites pour étudier les interactions existantes entre les aspects relationnels et temporels. Le monde abstrait que nous considérons sera alors un monde statique dans le temps en dehors des actions ou des robots et de leurs effets sur le monde. Notre planificateur ne devra pas être utilisé dans des missions où les aspects temporels sont importants.

Le propos de cette thèse est de proposer un planificateur raisonnant sur un modèle regroupant des connaissances géométriques, relationnelles et dans une moindre mesures de ressources. Nous verrons dans la suite que cette problématique nous amènera à lier la planification de tâches et la planification de mouvement. Nous unifierons ces deux domaines.

Pour l'heure, pour que le lecteur comprenne rapidement quel type de problème notre planificateur devra résoudre, nous présentons ci-après quelques exemples de modèles auxquels nous nous intéressons.

I.2 Problèmes

Les problèmes sont classés par ordre de “difficulté apparente”. Nous y ferons souvent référence au cours de cette thèse. Toutefois, la plupart des définitions et des algorithmes seront illustrés avec le deuxième problème présenté ici : le problème interrupteur/radio.

Pour chacun de ces problèmes, nous donnons une description générale et nous listons de manière informelle les contraintes géométriques et les contraintes relationnelles principales qu'ils contiennent. Tous ces problèmes contiennent des contraintes liées à la cinématiques des robots et à la forme des environnements. Une solution à ces problèmes ne doit pas faire apparaître de collision.

I.2.1 Les chariots élévateurs

Description générale Un chariot élévateur robotisé doit déplacer une boîte d'une configuration initiale à une configuration finale. Les configurations initiales et finales du chariot et de la boîte peuvent être placées partout dans l'environnement statique. Nous pouvons faire évoluer le nombre de chariots élévateurs et le nombre de boîtes à déplacer. La figure I.1 illustre ce problème pour un chariot et une boîte ainsi que pour deux chariots et deux boîtes.

Aspects relationnels : Il n'y a pas d'aspect relationnel dans ces classes de problèmes.

Aspects géométriques : Les prises de boîtes par les chariots élévateurs sont discrètes. Ils peuvent saisir les boîtes selon quatre orientations différentes. L'orientation des boîtes est prise en compte, ceci est représenté par une flèche sur les boîtes.

I.2.2 Le problème interrupteur/radio

Description générale Ce problème met en jeu deux robots : `forklift` (le même chariot élévateur que précédemment) et `armbot` (un robot avec un bras) et un objet : `cbox` (une boîte communicante). Seul `forklift` peut manipuler `cbox`. La mission pour ces deux robots est de recevoir un jeu de données et de déplacer `cbox` de l'autre côté de l'environnement. Les données peuvent être reçues par ondes radios et la réception n'est bonne que dans une aire spécifique de l'environnement. `cbox` est équipé d'une antenne adaptée pour recevoir de telles données. `forklift` doit transporter `cbox` pour recevoir ces données. Les données sont envoyées de manière continue après que l'interrupteur a été appuyé. Seul `armbot` est capable d'appuyer sur l'interrupteur. Ce problème est illustré sur la figure I.2.

Aspects relationnels : 3 relations de haut niveau[1]existent dans ce problème :

[1]Dans cette thèse, nous dénoterons par “haut niveau”, les relations, les tâches, les faits qui ne mettent

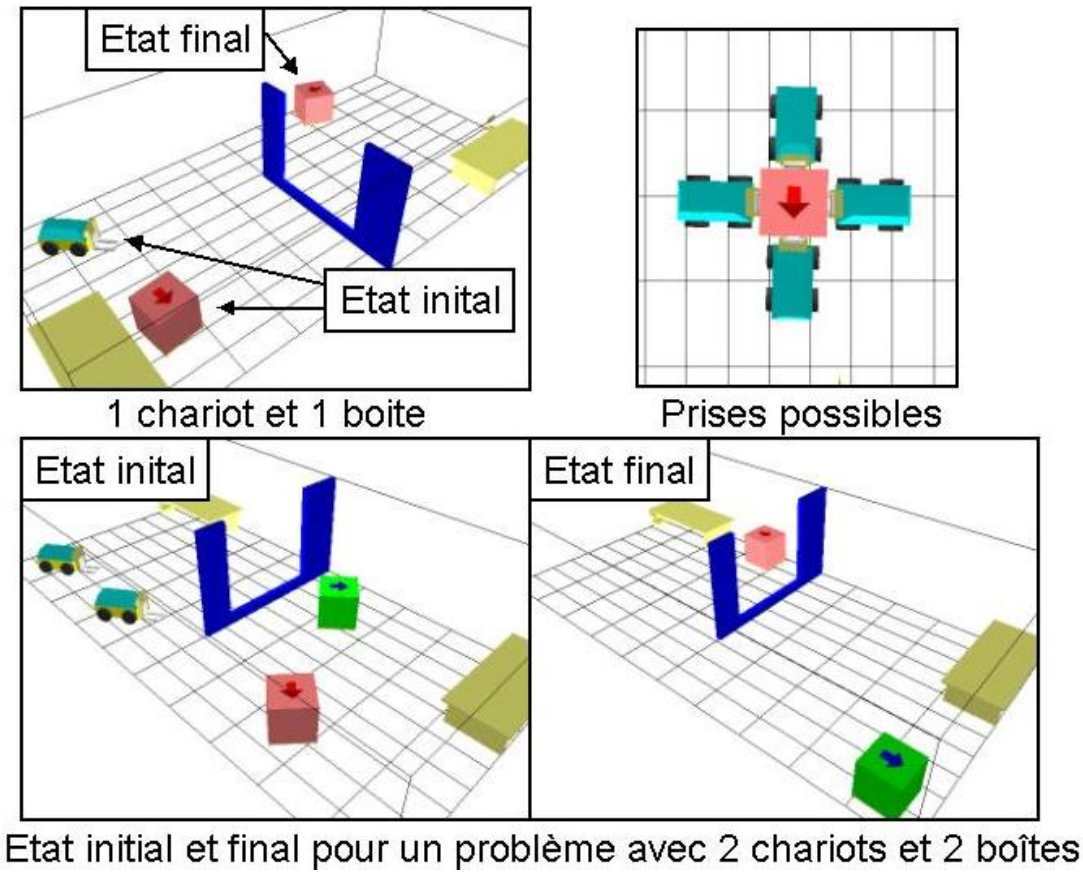

FIG. I.1 – Exemples de problèmes de chariots élévateurs.

– La relation entre le fait d'appuyer sur l'interrupteur et la présence d'un flux de données,

– La relation entre l'aire de "bonne réception" et la possibilité de recevoir des données.

– La relation entre le fait de transporter `cbox` et la possibilité de recevoir les données.

Aspects géométriques : Même type de prises que pour les problèmes précédents. On peut également remarquer que `armbot` est holonome[2] mais pas `forklift`, ils ont

pas en jeu de contraintes liées aux mouvements ou au manipulations

[2]la propriété intéressante pour les roboticiens d'un robot holonome est que le nombre de variables contrôlables est même que le nombre de dégrés de liberté du robot. Pour un système non-holonome, le nombre de variables controlables est inférieur au nombre de degré de liberté.

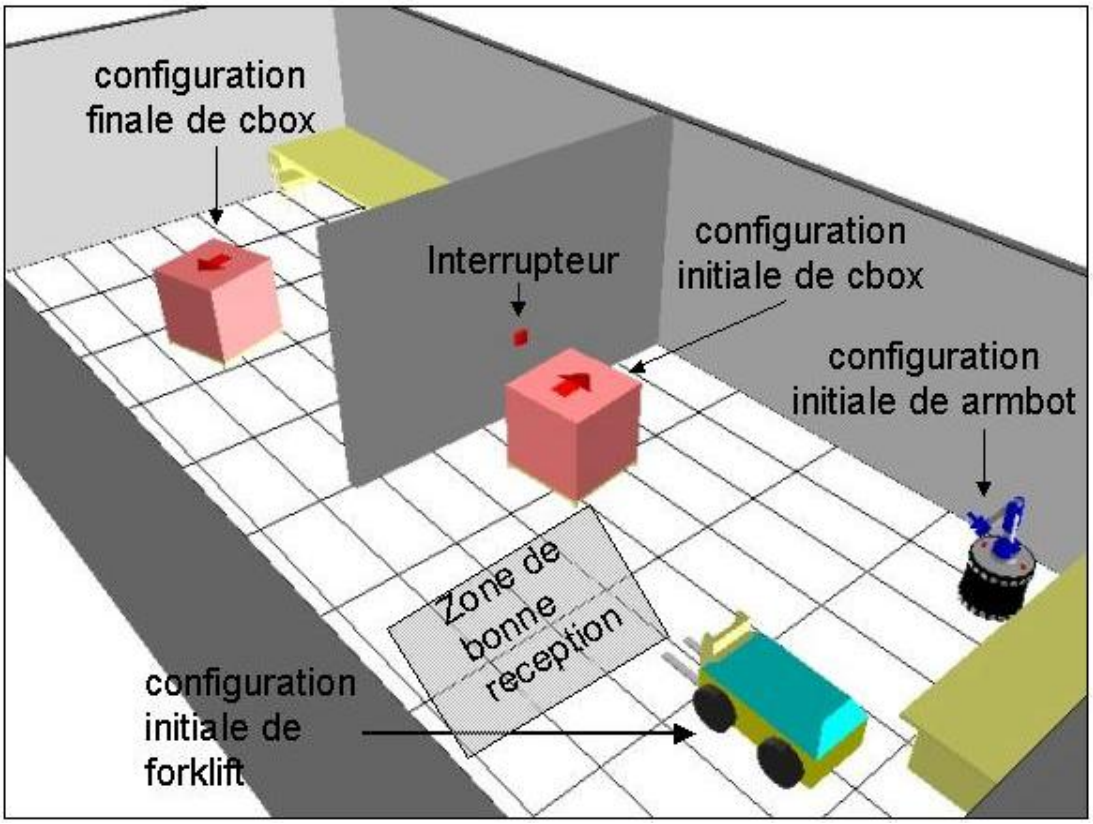

FIG. I.2 – Le problème interrupteur/radio.

des cinématique différentes.

I.2.3 Les problèmes des tours de Hanoï

Description générale : le problème des tours de Hanoï est directement inspiré du fameux jeu inventé par Edouard Lucas maintes fois résolu par les algorithmes d'Intelligence Artificielle. Le jeu consiste à recomposer une tour de trois cylindres de la pile de droite à la pile de gauche. Il est possible d'utiliser la pile centrale pour y déposer les cylindres en position intermédiaire. Les deux contraintes de ce jeu sont : premièrement, on ne peut déplacer qu'un seul cylindre à la fois, deuxièmement, on ne peut pas placer un cylindre sur un autre de plus petite dimension. Nous proposons quatre variantes de ce problème visible sur la figure I.3. Les variantes 1 et 2 se déroulent dans un environnement statique vide mais la variante 2 met en jeu deux robots chariots élévateurs. Les

variantes 3 et 4 ne mettent en jeu qu'un seul robot mais on modifie l'environnement statique en ajoutant un obstacle central plus ou moins volumineux.

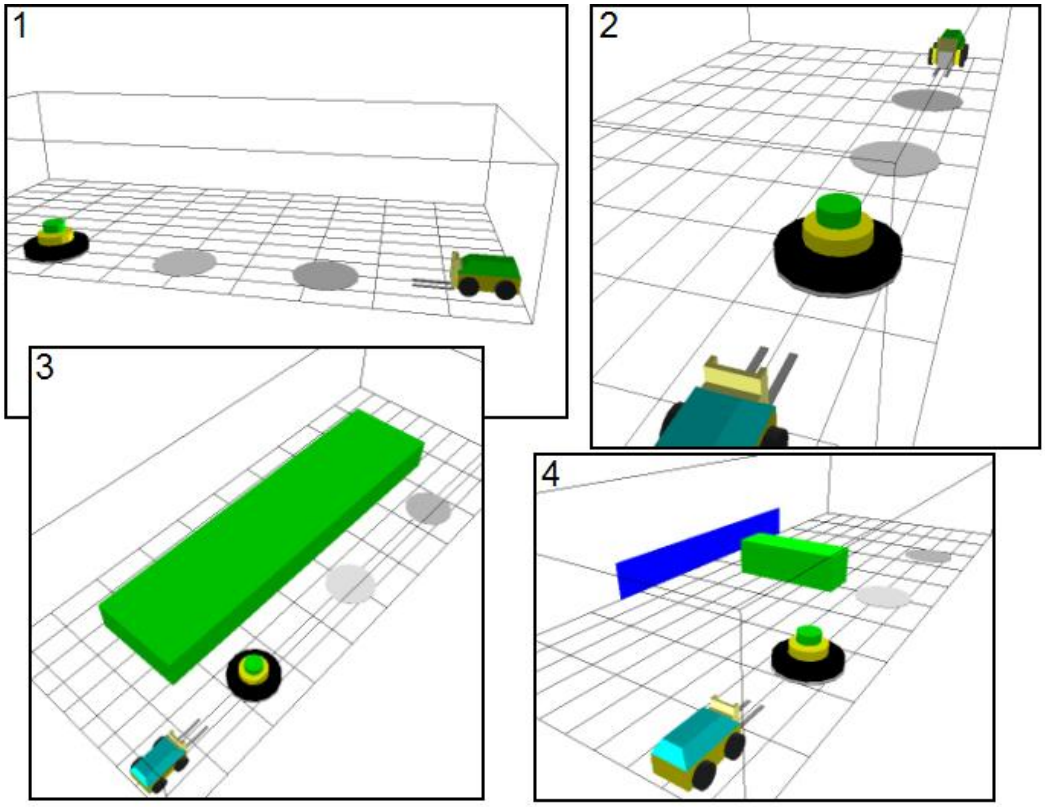

FIG. I.3 – Quatre instances du domaine des tours de Hanoï

Aspects relationnels : des relations existent entre les cylindres, leurs dimensions et les endroits où ils peuvent être déposés.

Aspects géométriques : les robots peuvent soulever les cylindres selon n'importe quelle orientation radiale. Selon la dimension de l'obstacle central et la position des objets, des piles seront inaccessible.

I.2.4 Le problème IKEA

Description générale : deux robots (`assembleur` et `colleur`) ont pour mission d'assembler une table en kit. Ils disposent pour cela de deux pieds de table à double tenants, de la planche et d'un distributeur de colle. Avant d'assembler un pied de table à

la planche, le robot `colleur` doit déposer deux points de colle sur la planche. Le robot `colleur` peut recharger sa réserve de colle à un distributeur. Initialement la réserve de colle du robot `colleur` est vide. Autre contrainte, il ne peut charger de la colle dans son réservoir de colle que pour appliquer deux points de colle, la construction de la table en nécessite quatre. Une fois la table assemblée, les deux robots doivent la retourner ensemble pour la poser dans sa position finale.

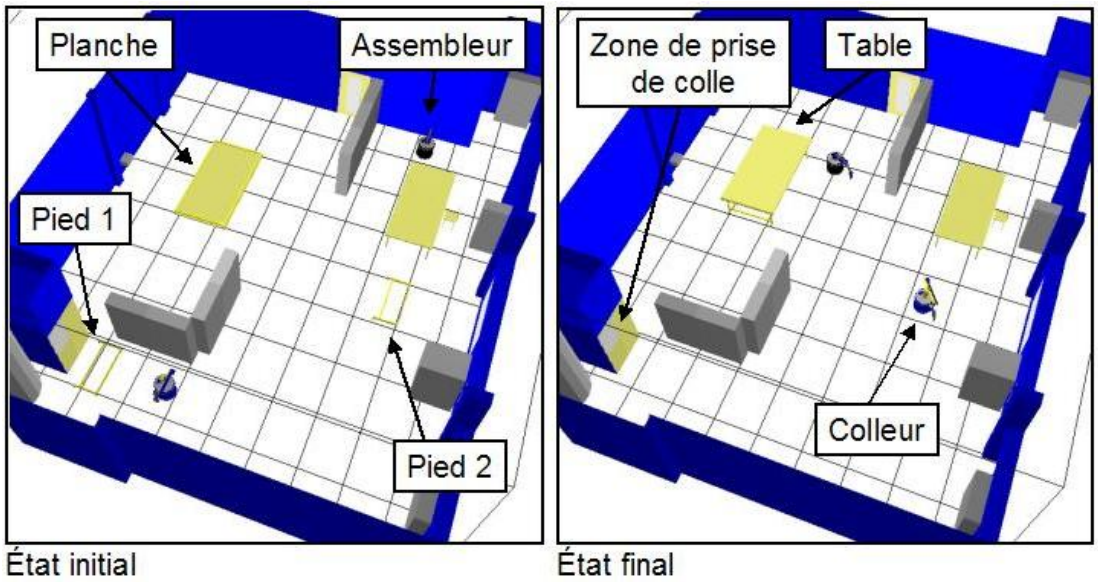

FIG. I.4 – Le problème “IKEA”.

Aspects relationnels : il y a des relations entres les objets en kit et leur procédure d’assemblage.

Aspects géométriques : les robots peuvent saisir les pieds, la planche ou la table assemblée en une infinité de prises possibles. Nous parlerons de *prises continues* pour ce cas de figure. Le pied 1 obstrue le passage menant au distributeur de colle.

Aspects de ressources : la réserve de colle du robot.

I.3 Plan de la thèse

Le lecteur doit avoir maintenant une idée précise des problèmes que nous souhaitons résoudre. Nous proposerons dans ce manuscrit une méthode de représentation ainsi qu’une méthode de résolution pour de tels problèmes.

Dans le chapitre suivant, nous parlerons brièvement des représentations utilisées en planification de tâches et en planification de mouvements. Nous argumenterons le fait qu'un système planificateur de tâches seul ou un système planificateur de mouvement seul ne puisse pas gérer aisément de tels problèmes. Nous écarterons également l'idée d'une hiérarchie simple entre deux de ces systèmes. Nous conclurons ce chapitre en proposant d'intégrer plus étroitement les deux types de planification.

Dans le troisième chapitre, nous étudierons les problèmes sous l'angle des espaces de configurations qu'ils mettent en jeu. Nous introduirons des relations externes capables de lier des symboles au monde géométrique. Ce paradigme nous permettra de définir formellement nos problèmes en gardant le découpage en domaine et en problème issu de la planification de tâches.

Le quatrième chapitre proposera une approche probabiliste d'exploration des espaces de configurations considérés. C'est une extension des méthodes à roadmaps probabilistes (PRM) pour les problèmes de manipulation multi-robots.

Le cinquième chapitre présentera l'algorithmique d'un nouveau planificateur nommé aSyMov capable de résoudre les problèmes proposés ici[3].

Le sixième chapitre montrera comment finalement nous arrivons à extraire un plan d'actions mêlant des mouvements des manipulations et des tâches de plus haut niveau. On montrera comment nous pouvons améliorer ce plan de manière conséquente.

Le septième chapitre est consacré à l'évaluation du système. On pourra en déduire les limites et les intérêts de celui-ci.

Le dernier chapitre est plus prospectif et fera office de conclusion. Nous y ferons une réflexion sur l'utilisation d'aSyMov pour des robots réels, et comment modifier ce système pour en tirer un maximum d'avantages.

[3]Le lecteur déjà curieux de visualiser les solutions trouvés sur les problèmes proposés ira télécharger les plans sous formes séquences mpeg sur http ://www.laas.fr/~scambon

Chapitre II

la planification de tâches et la planification de mouvements

Les problèmes précédents font appel à deux domaines de recherche : la planification de tâches et la planification de mouvements. Dans ce chapitre introductif, nous introduirons sommairement ces deux domaines. Nous argumenterons pourquoi les représentations utilisées par ces deux domaines ne peuvent suffire seules à représenter nos problèmes. Nous introduirons alors l'idée d'une intégration entre deux systèmes hétérogènes pour les résoudre.

II.1 La planification de tâches

La planification de tâches est un processus de raisonnement. C'est un des domaines de recherche de l'Intelligence Artificielle. Elle permet entre autre à des artefacts de se doter de capacités de décision sur les actions qu'ils sont capables de réaliser. Cette section n'a pas pour but de faire un état de l'art de la planification de tâches, nous renvoyons le lecteur à un livre récent [Ghallab 04].

La planification de tâches repose sur une représentation abstraite de l'environnement qui regroupe l'état du monde dans lequel évoluent les artefacts et l'état des artefacts dans cet environnement. On se munit également d'une représentation abstraite

des actions que les artefacts peuvent appliquer dans certains états du monde et de leurs conséquences sur le monde abstrait.

Un planificateur de tâches est un système capable de raisonner automatiquement sur ces représentations. En lui donnant un état initial de l'environnement et un ensemble d'états finaux de l'environnement, un tel système est capable de fournir un ordre d'actions permettant d'atteindre un des états finaux à partir de l'état initial.

Le formalisme STRIPS [Fikes 71] modélise les états par une conjonction de faits réduite grâce à l'utilisation de l'hypothèse du monde clos (seuls les littéraux positifs sont conservés dans la description de l'état). Les actions elles sont représentées par trois conjonctions de faits : les préconditions, les effets positifs et les effets négatifs. Les préconditions représentent ce qui doit être vrai dans l'état pour que l'action soit *applicable*. Les effets positifs représentent ce qui devient vrai après l'application de l'action, les effets négatifs ce qui devient faux. Depuis STRIPS, les problèmes de planification s'expriment avec deux entrées :

– Le *domaine*, il recense les actions souvent dans une forme non instanciée. On les appelle alors *opérateurs*.

– Le *problème* explicite lui l'état initial et une spécification des états finaux. Les symboles définis ici permettront de produire des actions (opérateurs instanciés).

Ce formalisme est puissant car il est capable de représenter les conditions d'application des actions et leurs conséquences sur le monde. Par contre il présuppose un monde statique, il n'y a pas de représentation explicite du temps, des ressources et des incertitudes.

Au cours des années qui ont suivi, de nouveaux systèmes planificateurs ont été créés pour gérer des représentations plus riches. Dans le but de comparer les algorithmes, la plupart des planificateurs utilisent désormais un langage commun : le PDDL (Planning Domain Definition Language) [Ghallab 98]. Plus récemment, ce langage a été étendu en PDDL2.1 pour gérer une représentation explicite du temps et des ressources [Long 03]. Il a été remarqué que sa puissance d'expressivité est équivalente à d'autres types populaires de représentation [Fox 04]. L'étude de l'expressivité du langage PDDL2.1 est donc suffisante pour savoir quels types de problème on peut réellement représenter et résoudre avec l'état de l'art de la panification de tâches.

II.1.1 Un planificateur de tâches pour résoudre nos problèmes ?

Nous voulons produire des plans qui assurent la faisabilité des mouvements en prenant en compte les contraintes issues de la géométrie du monde et de la cinématique des systèmes. Le plan doit assurer que les systèmes mobiles et le monde géométriques ne rentrent pas en collisions. Nous nous intéressons également à la manipulation d'objet, ce qui nous pousse à avoir une représentation fine et tridimensionnelle de la géométrie

du monde représentée de manière numérique.

Le langage PDDL2.1 permet de représenter des quantités numériques qui sont alors liées à des faits. On peut utiliser des prédicats arithmétiques pour comparer ces quantités. Dans les effets des actions on peut assigner de nouvelles valeurs aux quantités numériques. Pour nos problèmes, nous pourrions imaginer de créer des littéraux pour chacun des segments composant une scène tridimensionnelle. Des actions représenteraient des mouvements élémentaires des systèmes. Elles contiendrait en précondition une liste exhaustive de comparaison entre tous les littéraux représentants la configuration d'arrivée du système mobile et les littéraux correspondants au segments de l'environnement statique. Ce mécanisme pourrait juger de la non-collision d'une action de mouvement élémentaire. Les effets de telles actions assigneraient une nouvelle valeur au positionnement des segments représentant le robot. Ce type d'approche paraît extrêmement lourde ne serait-ce que pour la spécification de tels domaines. De plus il est difficile d'imaginer un algorithme efficace pour résoudre un problème de planification ainsi exprimé. En effet, une discrétisation fine des mouvements en actions élémentaires augmente considérablement le nombre d'action à considérer même pour un problème simple.

Plus classiquement, les planificateurs de tâches délèguent l'expertise sur le mouvement à un unique prédicat. Nous donnons ci-après l'exemple une action représentant un mouvement pour un rover d'exploration planétaire. Cette action est issue d'un domaine classique de la planification de tâches utilisé lors des compétitions d'algorithmes planificateurs [Cambon 02].

```
(:action navigate
 :parameters (?x - rover ?y - waypoint ?z - waypoint)
 :precondition (and   (can_traverse ?x ?y ?z)
                              (available ?x)
                               (at ?x ?y)
                      (visible ?y ?z))
 :effect       (and   (not (at ?x ?y))
                              (at ?x ?z))
)
```

Le prédicat `can_traverse` représente ici la faisabilité de mouvements pour le rover. On peut imaginer que pour ce type d'application, un planificateur de chemins a été appelé en prétraitement avant de concevoir le problème de planification.

Dans notre cas, plusieurs robots hétérogènes peuvent évoluer dans l'environnement et peuvent manipuler des objets. Le prédicat `can_traverse` devra avoir des valeurs de vérité différentes selon :

– Le robot. Un robot peut ne pas passer là où un robot de forme différente le peut.

– Que le robot transporte un objet ou non. Un robot transportant un objet a une forme différente, on se ramène donc au premier cas.

– La position des objets déplaçables. Le déplacement d'un objet implique des changements de la géométrie du monde, et peut donc libérer ou obstruer des passages.

Avec ce genre de représentation, si on considère 2 robots, 2 objets qu'ils peuvent déplacer et 10 "*waypoints*" le nombre d'instance du prédicat `can_traverse` à évaluer dépasse les 15000. Cette approche devient même complètement hors d'usage si nous considérons que les placements possibles pour les objets et les robots sont continus.

Les paradigmes de la planification de tâches nous semblent inadaptés aux calculs des mouvements ou la prise en compte d'une géométrie complexe.

II.2 La planification de mouvements

La planification de mouvements peut être vue comme un processus de raisonnement artificiel particulier. Le problème de la planification de mouvement est de trouver un mouvement pour un artefact permettant d'atteindre une configuration finale à partir d'une configuration initiale. Ce domaine de recherche est souvent rattaché à celui de la Robotique. Dans cette partie nous ne parlerons que du problème fondateur de la planification de mouvement, celui d'un système articulé unique ou considéré comme unique évoluant dans un environnement statique.

Depuis Lozano-Perez [Lozano-Pérez 83], le problème est formulé comme une recherche dans l'espace des configurations *CS*. Une configuration est une instance des degrés de liberté de l'artefact considéré. Un mouvement est donc une courbe (appelé *chemin*) dans l'espace des configurations *CS*. L'espace *CS* est de dimension égale au nombre de degré de liberté du système considéré.

Il a été montré que la planification d'un mouvement pour un mécanisme articulé composé de polyèdres dans un environnement tridimensionnel est PSPACE-difficile [Reif 79]. L'algorithme exact (i.e. une approche *complète* qui garantit une solution si elle existe et qui retourne un échec si le problème n'est pas résoluble) le plus efficace est d'une complexité en temps exponentielle selon la dimension de l'espace des configurations [Canny 88].

Conséquemment, les recherches effectuées en planification de mouvement cherchent à casser cette complexité. Les premières approches ont été de discrétiser certaines valeurs continues comme les dimensions des objets ou les paramètres des configurations. On décompose alors l'espace libre de *CS* (CS_{free}) en cellule. On peut également se ramener à des grilles. On peut également évoquer l'approche par champs de potentiel qui est rapide mais peut être facilement piégée dans des minimums locaux. Toutes ces ap-

proches sont clairement détaillées dans le livre référence de Latombe ([Latombe 91]).

Le défaut principal des approches utilisant des cellules ou des grilles est qu'elles sont applicables en pratique que sur des systèmes mobiles mettant en jeu peu de variables. Pour des dimensions plus élevées, comme dans la plupart des systèmes réels, le nombre de cellules ou de points dans la grille devient trop important. Malgré tout, les algorithmes basés sur ces représentations sont complets pour un pas de discrétisation donné. Pour l'approche par champs de potentiel, le point critique est la conception de la fonction de potentiel pour les espaces hautement dimensionés. Cette conception n'est pas aisée pour les systèmes ayant un grand nombre de degré de liberté.

Au début des années 90, un nouveau genre de planificateur de mouvement est apparu basé sur des approches stochastiques. L'avantage de ces méthodes est de pouvoir planifier des mouvements pour des systèmes ayant un grand nombre de degrés de liberté. Vu que nous nous intéressons à la manipulation, les systèmes mécaniques considérés ont souvent cette propriété. Par exemple un bras manipulateur monté sur une base mobile possède 9 degrés de libertés. Pour cette raison, nous ne nous intéresserons plus qu'à ce genre de planificateur de mouvements au cours de cette thèse. Ils sont brièvement présentés ci-après.

II.2.1 Approches probabilistes

Les approches probabilistes consistent à caractériser l'espace libre ou une portion de celui-ci à l'aide de graphes et de manière stochastique [Barraquand 91] [Kavraki 96] [LaValle 98].

Dans ces graphes, les noeuds sont des configurations et les arêtes, appelé chemins locaux, représentent des mouvements valides et sans collisions entre deux configurations. Les configurations sont tirées au hasard puis reliées entre elles.

Deux composants logiciels sont essentiels à la construction de ces graphes :

– **La méthode locale**. Elle permet de savoir si deux configurations peuvent être reliées par un mouvement en respectant les contraintes cinématiques du système étudié et la limite des joints des articulations sans se préoccuper des obstacles. Pour les systèmes mobiles holonomes, elle est facile à calculer. Une variation linéaire de chacun de degré de liberté de la configuration initiale à la configuration finale est admissible. Cela correspond à une ligne droite dans *CS*. Pour les systèmes non-holonomes la conception d'une méthode locale est un problème difficile.

– **Le détecteur de collisions**. La géométrie euclidienne montre ici ses limites. Le système mobile le long d'un chemin local forme une "soupe" de polygone difficile à mettre en équations. Pour savoir si ce chemin rentre en collision avec l'environnement géométrique, on fait alors appel à des approches "informaticiennes" où le chemin local est discrétisé. Chacune des configurations obtenues doivent alors être

testées grâce à un détecteur de collision statique qui met en oeuvre un mélange de géométrie euclidienne et de méthodes calculatoires. Le détecteur de collisions est le composant le plus intensivement appelé par les planificateurs de mouvements stochastiques, son optimisation est donc importante.

D'autres composants peuvent être cruciaux pour les performances des planificateurs de mouvements à approches probabilistes comme par exemple la stratégie de connexions ou le tirage aléatoire des configurations qui peuvent trouver avantage à ne pas être totalement aléatoire justement. Nous ne développerons pas ces aspects ici.

Parmi ces approches probabilistes, on peut distinguer deux types d'approches. Nous les présentons ici et fournissons des exemples d'algorithmes. Ces exemples ne sont pas choisis au hasard dans la mesure où ils sont réellement implantés sur le planificateur de mouvement Move3D [Siméon 01] que nous utiliserons.

Les **Probabilistic Roadmap Method** ou **PRM**. Ces méthodes cherchent à capturer CS_{free} dans son ensemble. Elles construisent un graphe nommé *roadmap*[1], ce graphe construit des "routes" que le système pourra employer pour rejoindre une configuration finale en partant d'une configuration initiale. Ces deux configurations seront données a posteriori. La PRM classique échantillonne des configurations que l'on lie via la méthode locale. La stratégie de connexion est souvent de tester d'abord les arêtes entre les noeuds les plus proches de chaque composante connexe avec la nouvelle configuration. Quelques méthodes permettent d'améliorer l'algorithme, comme par exemple la **Visibilité PRM** [Nissoux 99]. Cette extension permet de limiter le nombre de noeud et d'obtenir une notion de topologie "suffisante". Lors du tirage d'une nouvelle configuration, on ne la gardera dans la roadmap que si elle permet de relier deux composantes connexes distinctes ou si la configuration (i.e. le noeud) est isolée. Lorsqu'un certain nombre d'échec d'insertion de nouveaux noeuds est constaté, on peut considérer que CS_{free} est suffisamment connu. Lors d'une requéte de calcul de mouvement les configurations initiales et finales de la requéte auront de forte probabilité de pouvoir être relié à la roadmap avec une seul arêtes.

L'approche PRM est connue pour être complète en probabilité. Une preuve générale de cette propriété pourra être trouvée dans [Kavraki 95]. Ceci est même étendu pour le cas plus général où les systèmes sont affectés par des contraintes non holonomes [Svestka 97]. C'est à dire que la chance de trouver une solution si elle existe tend vers 1 lorsque le temps imparti à la résolution tend vers l'infini. En d'autres termes, lorsque l'on utilise un planificateur de mouvement de ce type, on ne peut jamais être totalement convaincu qu'un problème est sans solution. Cela aura de l'importance dans la suite.

La **recherche incrémentale** est la seconde approche. On ne cherche plus ici à capturer la topologie CS_{free} dans son ensemble mais un sous-espace de celui-ci cor-

[1] "carte" ou "feuille de route", dans cette thèse nous garderons la dénomination anglaise "roadmap" considérée comme un nom fémin.

respondant à la requête contenant une configuration initiale et une configuration finale. On ne cherchera alors à caractériser que la composante connexe regroupant la configuration initiale et la configuration finale. Dans ces méthodes, l'une des plus populaire est le Rapidly-exporing Random Tree ou RRT. Comme son nom l'indique, le graphe construit sera un arbre dont la racine est la configuration initiale. Lors du tirage d'une nouvelle configuration q, le noeud de l'arbre le plus proche selon une métrique donnée est étendu vers q. Dans la version basique du RRT (ou RRT-extend), l'extension consite en un ajout, si c'est possible, d'une seule nouvelle configuration à une distance donnée (de la même métrique donnée) vers la direction de q. Dans une version plus élaborée (RRT-connect), on ajoutera autant de configuration que possible espacé de la même distance métrique vers q. On peut également faire une recherche bidirectionnelle en construisant deux arbres, un partant de la configuration initiale l'autre partant de la configuration finale. Cette approche est plus utilisée dans le cas d'une requête unique d'un calcul de mouvement.

II.2.2 Un planificateur de mouvement pour résoudre nos problèmes ?

Les planificateurs de mouvements travaillent sur une représentation géométrique du monde. Le monde est décrit sous forme de polygones statiques ou pouvant se déplacer selon des contraintes clairement définies. Dans les problèmes que nous voulons résoudre, nous avons besoin de représenter des relations non géométrique entre les objets géométriques du monde ou même entre des propriétés de l'environnement que l'on ne peut pas représenter par la géométrie.

Par exemple, dans le problème des tours de Hanoï (cf. I.2.3), les robots ne peuvent pas empiler un disque sur un autre plus petit. Nous pourrions gérer cette contrainte en énumérant toutes les configurations impossibles. Malheureusement, à chaque changement, même infime, de problème (augmentation du nombre de disques, déplacement des piles...), il faut revoir cette liste de configurations interdites.

Dans le problème IKEA (cf. I.2.4), les pieds de tables peuvent être assemblés après que le plateau ait reçu deux points de colle. Il faut donc que le robot colleur ait visité une configuration correspondant à la recharge de colle et deux configurations correspondant à la pose de points de colle avant que l'assemblage ne puisse être réalisé.

L'utilisation d'un planificateur de mouvement semble possible mais nécessite une introduction de contraintes ad hoc pour chaque problème. De plus ces contraintes ne peuvent être définies que pour des configurations précises et doivent être revues à chaque changement même infime des données d'entrée du problème. Une représentation symbolique de ces contraintes est bien plus naturelle et reste indépendante du contexte géométrique précis. Ce sera le choix que nous ferons dans cette thèse. Nous verrons que nous lierons le monde géométrique à des symboles à travers des relations externes.

II.3 Intégrer la planification de tâches et la planification de mouvements

Comme nous venons de le voir la planification de tâches telle qu'elle est étudiée ces dernières années n'est pas appropriée pour planifier des mouvements complexes. Les représentations des planificateurs de tâches n'étant pas adaptées à la représentation d'un monde continu tridimensionnel complexe. Les concepts de la planification de mouvements, eux sont adaptés à ces problèmes mais ne permettent pas de représenter aisément des relations non géométriques. La question centrale de cette thèse est de savoir comment regrouper ces deux types de représentations et comment par la suite explorer l'espace qui sera ainsi formé pour trouver des solutions.

II.3.1 Une hiérarchie de planificateur ?

L'idée relativement naturelle d'une hiérarchie composée de haut en bas d'un planificateur de tâches et d'un planificateur de mouvement est une mauvaise idée. Cela doit être clair dans l'esprit du lecteur, car il sous-tend cette présente thèse. Jean-Claude Latombe dans son livre de référence "Robot Motion Planning" de 1991 [Latombe 91] écrivait :

"[...] a plan generated by the tasks planner may turn out to be infeasible when developed at the motion level.[...] In such a situation, the taks planner would have to generate another plan. Important questions are : What kind of pertinent information should the motion planner feed back to the task planner ? How can the task planner use this information to generate another plan ?[2]

L'auteur précisait dans le même paragraphe que les avancées dans le domaine d'une bonne intégration entre la planification de tâche et la planification de mouvements était presque nulles. Force est de constater qu'il en quasiment de même aujourd'hui. C'est clairement le sujet de cette présente thèse fruit de la collaboration avec Fabien Gravot ([Gravot 04] : pour consulter ses travaux de thèse) et Rachid Alami.

Pour écarter définitivement l'idée d'une hiérarchie simple entre un planificateur et un sous-système, nous pensons que celle ci ne peut fonctionner correctement que si les résultats des calculs du sous-système n'ont aucune conséquence sur la représentation du monde du planificateur. Autrement dit les calculs du sous-système doivent être faits à un moment dans le plan mais les résultats de ces calculs ne remettent pas en cause le

[2][...] un plan généré par un planificateur de tâches peut devenir infaisable quand il est développé au niveau des mouvements. [...] Dans une telle situation, le planificateur de tâche devrait générer un autre plan. Les questions importantes sont : Quels types d'informations pertinentes le planificateur de mouvement doit renvoyer au planificateur de tâche ? Comment le planificateur de tâche utilise-t-il ces informations pour générer un autre plan ?

plan de haut niveau. Nous ne pouvons clairement pas procéder ainsi pour le mouvement dans le cadre de la robotique mobile. En effet la faisabilité d'une action dépend presque toujours de la possibilité de rejoindre une configuration permettant d'appliquer cette action. En robotique mobile, la faisabilité des mouvements sont des préconditions aux actions.

Il existe toutefois une hiérarchie simple qui fonctionnerait correctement et que n'évoque pas Latombe. C'est tout simplement la hiérarchie inverse : on fait appel à un planificateur de mouvement puis à un planificateur de tâche. Le planificateur de mouvements devra alors attribuer à tous les mouvements possibles une valeur de vérité. Cette approche est déjà critiquée plus haut : le nombre de littéraux évaluer exploserait. Ce type d'approche ne nous paraît pas atteignable.

Il existe dans la littérature des exemples d'interaction un peu plus complexe d'un planificateur de tâches et un ou plusieurs sous-systèmes. Ce genre d'intégration peut largement profiter de planificateur maniant des hiérarchies d'abstraction comme IxTeT [Laborie 96] ou d'une représentation de type HTN comme SHOP [Nau 03]. Dans les deux cas on peut clairement séparer la résolution des évolutions des attributs et donc de faire appel à un sous-système spécialisé dans cette résolution en cours de planification et non plus après le calcul d'un plan de haut niveau. Par exemple l'évolution de la position d'un robot dans son environnement peut être résolu après avoir calculer l'ordonnancement des tâches permettant de déterminer l'évolution des attributs de "plus haut niveau". Ce sont souvent des attributs lié à la mission du robot (distribution de courrier, surveillance, exploration...).

Par exemple, des travaux ont été réalisé pour lier IxTeT avec un planificateur d'itinéraires [Lamare 98]. Dans ces travaux, toute la connaissance sur les lieux connus du robot est regroupée dans un graphe. Les itinéraires du robot sont calculés par le planificateur secondaire en cours de planification. Malheureusement, dans nos problèmes cela reviendrait une nouvelle fois à faire appel à une "hiérarchie inverse" : tous les mouvements doivent être calculés au préalable.

II.3.2 L'intégration des tâches, des mouvements et des manipulations dans l'état de l'art

L'idée d'une hiérarchie entre deux planificateurs étant désormais écartée, nous donnons ici des exemples dans la littérature de systèmes où les interactions entre les tâches, les mouvements et les manipulations sont explicitement prises en compte.

Un des points clé que nous voulons traiter dans cette thèse est celui de la manipulation. Lorsque l'on prend en compte cet aspect, un lien fort entre la géométrie et les taches apparaît. Par exemple selon la prise choisie pour saisir un objet, il pourra être déposé ou non sur une table. On ne peut pas alors dissocier aisément la tâche "déposer

sur la table" de la prise à choisir. Des travaux ont été produits dans les années 80 sur ce problème de programmation de robot. Nous pouvons par exemple citer les systèmes HANDEY [Lozano-Perez 87], SHARP [Laugier 85] ou encore SPARA [Mazon 90]. Ces systèmes ne bénéficiaient pas de l'apport des approches probabilistes pour la planification de mouvements. Cette thèse peut être vue comme une mise à jour de ces idées un peu délaissées depuis.

Tous ces systèmes ont été conçu pour le problème de la planification d'assemblage. Cet assemblage pouvant être effectué par une cellule robotisée. Une bibliographie assez complète des recherches avant 1993 (la période des années 80 aux début des années 90 est la plus prolifique sur ce problème) sur ce problème peut être trouvée en [Gottschlich 93]. Dans ce type de problème, des contraintes relationnelles et des contraintes géométriques non indépendantes apparaissent. Le problème de la planification d'assemblage est le problème le plus proche du notre. La différence avec notre problématique est que nous travaillons dans le cadre de la robotique mobile et donc les objets et les robots ne sont pas limité en placement dans le monde géométrique. La différence peut paraître minime mais elle invalide à notre connaissance la plupart des systèmes issus de ces recherches pour résoudre notre problématique. En effet dans ses systèmes, les contraintes géométriques existant entre les placement des objets, les prises possibles et les contraintes d'assemblage sont exhaustivement recensées dans un graphe [Halperin 98] ou avec des contraintes [**?**]. En robotique mobile, si on considère que les robots et les objets peuvent être placés n'importe où dans l'environnement, ce recensement n'est pas atteignable.

Par exemple dans [Hutchinson 90], un des travaux les plus proches des nôtres, les auteurs étendent la planification de tâches pour ce problème. Les préconditions des tâches sont divisées en trois types. Des préconditions opérationnelles, qui sont des préconditions "à la STRIPS" comme par exemple la fait que pour qu'un bras manipulateur puisse saisir un objet il faut que la pince du bras soit libre et ouverte. Des préconditions géométriques qui garantissent par exemple que la pince peut atteindre l'objet. Les auteurs ont également rajouté des préconditions liées aux incertitudes, nous ne les détaillons pas ici. La gestion des contraintes géométriques est très simplifiées : ce sont un ensemble de contraintes binaires entre les orientations des objets et les prises possibles sur ces objets. Dans le cas de la robotique mobile, cette gestion est trop simple car il paraît difficile de recenser complètement toutes ces contraintes. L'infinité de placement des objets et des robots dans l'environnement en est la cause.

Pour ces raisons, notre système ne fera pas de recensement de toutes les contraintes pouvant apparaître au niveau géométriques. La découverte de celles-ci se fera en cours de planification.

D'un point de vue général, nous utiliserons une approche proche de celle suggéré par Latombe dans la section précédente. Celle-ci suppose que l'on tente de vérifier

qu'un plan de tâche est réalisable au niveau des mouvements. Si ce n'est pas le cas, le planificateur de tâche proposera un nouveau plan sur la base d'un bilan renvoyé par le planificateur de mouvements. Il y a pourtant certains détails qui ont leurs importances dans la mise en place de notre approche. En effet, nous avons décidé de nous reposer sur des approches probabilistes qui ont les propriétés suivantes :

– (1) Elles ne garantissent pas l'existence d'une solution en temps fini. On ne pourra donc jamais faire une affirmation du type "ce mouvement est impossible".

– (2) Le fait de chercher une solution à un mouvement peut aider la recherche de solution d'un autre mouvement.

Notre approche fera donc une recherche du plan solution en explorant un espace de plans symboliques et en cherchant à trouver des solutions pour les mouvements qu'ils mettent en oeuvre. Les problèmes de planification de mouvement à résoudre seront choisis en fonction de l'intérêt symbolique qu'ils présentent et d'une mesure de l'espérance de réussite de trouver une solution à ce mouvement, mesure que l'on extraira de certains paramètres issus du planificateur de mouvement. Cette approche permet d'éviter de passer trop de temps dans le calcul d'un mouvement qui n'a pas de solution (première propriété) et de profiter de la recherche d'un mouvement pour permettre la recherche de solution d'un autre mouvement sur un autre plan (deuxième propriété).

Dans le chapitre suivant, nous posons formellement le type de problème que nous voulons résoudre.

Chapitre III

De la manipulation aux symboles : représentation des problèmes

Dans le chapitre précédent, nous avons mis en évidence les difficultés à représenter le mouvement et la manipulation dans une approche de planification de tâches classique. Toutefois les représentations sur lesquelles est basée cette planification permettent de raisonner facilement sur les relations de haut niveau pouvant exister entre les diverses parties d'un environnement. D'un autre côté, le champ de la planification de mouvements est spécialisé dans le calcul des déplacements valides des artefacts. Malheureusement la représentation géométrique sur laquelle est basée la planification de mouvement ne permet pas de représenter facilement les contraintes relationnelles de haut niveau.

Ce chapitre, central dans cette thèse, propose une mise en relation des représentations géométriques (les lieux, les robots, les objets) avec des représentations symboliques. Pour cela nous remarquerons tout d'abord que les contraintes de la manipulation que nous nous donnerons induisent une structure particulière dans l'espace des configurations global *CS*. Nous montrerons ensuite comment prendre en compte ces contraintes issues de la manipulation à un niveau symbolique. Nous procéderons en utilisant des relations externes liant les entités géométriques (sous-espace de *CS*, robots, objets) avec des symboles. Finalement, nous proposerons une définition formelle du problème de planification dans son ensemble.

III.1 Structure de l'espace des configurations pour un problème de manipulation multi-robots

La planification de manipulation est un champ de recherche visant à synthétiser automatiquement les séquences de mouvements de robot pour manipuler des objets dans un environnement statique. La présence d'objets mobiles, déplaçables seulement par les robots, nous place dans un cas plus général et plus complexe en terme de calcul que la planification de mouvements classique.

Le développement de l'approche que nous allons utiliser pour les problèmes de manipulation repose sur les articles [Alami 89], [Alami 95] et les travaux de thèse [Sahbani 03] et [Cortés 03]. Le tout est remarquablement formalisé et clarifié dans [Siméon 03]. Il est d'ailleurs à noter que les notations et les explications présentées ici sont largement inspirées de cette dernière citation. C'est également un des principaux travaux inspirateurs de cette thèse. Cet article définit notamment les sous-espaces de configurations remarquables de CS dans le cas d'un seul robot et d'un seul objet déplaçable. Nous étendons ici ces définitions dans le cas général de plusieurs robots et de plusieurs objets. Nous appelons ce problème le problème de manipulation multi-robots.

Plus formellement, dans notre espace de travail en trois dimensions, nous avons r robots R_i possédant respectivement n_i degrés de liberté et m objets rigides (i.e. sans articulations) M_i possédant chacun 6 degrés de libertés. Nous supposons être dans le cas général où tous les robots peuvent manipuler tous les objets. Nous prenons également pour hypothèse que les robots ne peuvent pas manipuler plus d'un objet à la fois. Cette hypothèse simplificatrice nous permet surtout de rendre les définitions plus simples, nous pourrions les étendre pour ce cas.

Soit CS_{R_i} et CS_{M_i} les espaces de configurations des robots et des objets respectivement. L'espace des configurations composite du système est $CS = CS_{R_1} \times ... \times CS_{R_r} \times CS_{M_1} \times ... \times CS_{M_m}$ et CS_{free} est le sous-espace libre de CS (i.e. l'ensemble des configurations sans collision de CS).

Les contraintes de la manipulation. Des contraintes sont définies par l'utilisateur. Les premières concernent l'ensemble des positions stables possibles pour les objets. Nous définissons un sous-espace de chaque $CS_{M_i} (0 < i \leq m)$ où les objets peuvent être déposés (pour satisfaire par exemple la gravité). Les secondes contraintes concernent les ensembles de positions de saisie des objets. Nous définissons pour chaque paire robot-objet un ensemble de configurations relatives de l'objet par rapport au robot qui correspondent à des prises valides. Les positions de prise doivent rester fixes lorsqu'un robot transporte un objet. Les positions de prises ainsi que les positions stables des objets peuvent être des ensembles continus. On peut remarquer que d'autres significations de la manipulation auraient pu être choisies comme pour le "poussé" des objets

qui introduirait d'autres contraintes spécifiques [Lynch 96] [Stilman 04].

De manière à simplifier le propos, nous ajoutons une dernière contrainte : il ne peut y avoir qu'un seul robot qui se déplace à la fois.

Maintenant nous considérons les sous-espaces de CS_{free} suivant :

– $CP(R_i)(0 < i \leq r)$, l'ensemble des configurations du robot R_i où celui-ci n'est pas attaché à un objet et où il peut se déplacer librement. Les autres objets sont dans une position stable ou sont attachés à un autre robot.

– $CP(M_i)(0 < i \leq m)$, l'ensemble des configurations de l'objet M_i où celui-ci est sur une position stable. Les autres objets sont également dans une position stable ou saisis par un robot.

– $CG(R_i.M_j)(0 < i \leq r)(0 < i \leq m)$[1], l'ensemble des configurations où le robot R_i saisit l'objet M_j et où les autres objets sont sur une position stable ou attrapés par un autre robot.

Une solution à un problème de manipulation multi-robots est un chemin dans CS_{free} respectant les contraintes de la manipulation. C'est une séquence finie de chemins de *transit-*R_i et de chemin de *transfert-*R_iM_j. Les chemins de transit et de transfert mettant en jeu le même robot sont séparés par des opérations de prise ou de pose.

Un **chemin de transit-**R_i est un chemin où le robot se meut seul et où les autres robots restent statiques. Ces chemins appartiennent à $CP(R_i)$. On peut remarquer que tous les chemins de $CP(R_i)$ ne sont pas forcément des chemins de transit. La plupart d'entre eux mettent en jeu des mouvements des autres robots ou des objets.

Un **chemin de transfert-**R_iM_j est un chemin où le robot R_i se déplace en maintenant une prise fixe de l'objet M_j. Les chemins de transfert appartiennent à $CG(R_i.M_j)$. On peut remarquer que, comme pour les chemins de transit, tous les chemins dans $CG(R_i.M_j)$ ne sont pas des chemins de transfert.

Nous remarquons également que les sous-espaces $CP(R_i) \cap CG(R_i.M_j)(0 < i \leq r)(0 < i \leq m)$ sont intéressants. En effet ils correspondent aux configurations où les chemins de transit R_i et les chemins de transfert $R_i.M_j$ peuvent être connectés par une opération de prise ou de pose.

Les différentes définitions faites jusque là sont illustrées sur la figure III.1 qui représente un problème de type "chariots élévateurs" (I.2.1). Contrairement à un problème de type "chariots élévateurs", nous n'autorisons que deux prises possibles de l'objet `cbox` par le chariot `forklift`. L'objet `cbox` doit être transféré. Sur la figure III.1 nous représentons les différents sous-espaces composant CS_{free}. On peut remarquer que $CP(forklift)$ et $CG(forklift.cbox)$ sont constitués de plusieurs composantes connexes distinctes. En conséquence, un chemin solution doit mettre en jeu deux opérations de prises de l'objet. `forklift` doit poser `box` sur une configuration intermédiaire où il

[1]Ici le symbole "." ne correspond pas à un opérateur, c'est une convention d'écriture permettant de séparer le noms du robots et de l'objet

peut changer de prise avec un chemin de transit appartenant à la composante connexe 3 avant d'amener `cbox` à sa configuration finale.

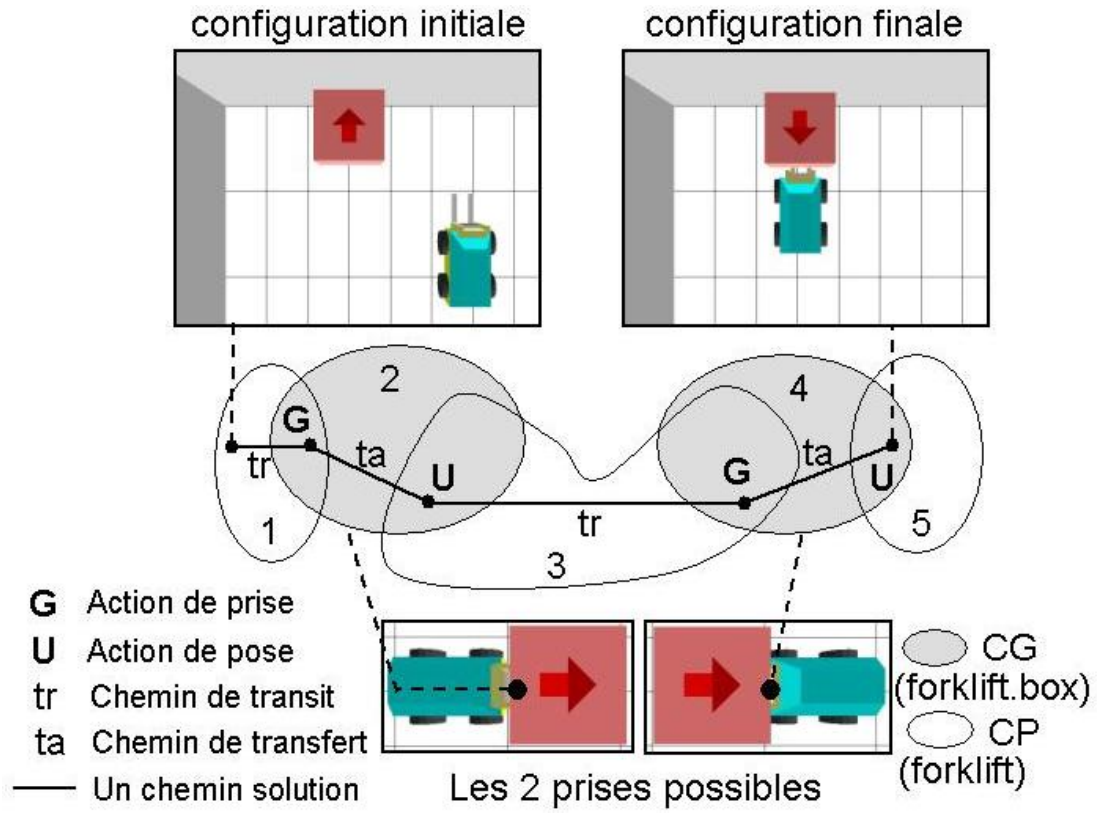

FIG. III.1 – Illustration des définitions sur la manipulations.

Une dernière remarque intuitive pour l'instant mais importante pour la suite du propos : les intersections entre les sous-espaces définis plus haut sont proches d'une notion d'état. Par exemple, $CP(R_1) \cap CP(R_2)$ est l'espace où le robot R_1 et le robot R_2 sont dans un état où ils ne sont attachés à aucun objet et où les autres robots et objets sont dans un état où ils sont dans une position stable. $CG(R_1.M_1) \cap CP(R_2)$ est l'espace où le robot R_1 attrape l'objet M_1 et le robot R_2 n'est attaché à aucun objet. Les autres robots et objets sont en position stable. Un dernier exemple $CG(R_1.M_1) \cap CG(R_2.M_1)$ est l'espace où le robot R_1 est attaché à l'objet M_1 et où le robot R_2 est également attaché à ce même objet.

Toutes ces intersections définissent un état précis du système. Nous verrons plus tard que nous attacherons à chacune de ces intersections une "étiquette symbolique" qui nous permettra de naviguer plus facilement dans CS_{free}.

Dans la section suivante, nous montrons comment nous relions les entités définies ici à des symboles et des prédicats qui seront utilisées par notre planificateur.

III.2 Une méthode de représentation symbolique des contraintes de la manipulation

Dans cette section, nous prendrons le problème interrupteur/radio (I.2.2) pour illustrer notre méthode.

Supposons que nous ayons une description géométrique du monde d'un environnement E composé d'obstacles statiques, d'un ensemble de robots R_i dont on décrit la géométrie et la cinématique, d'un ensemble d'objet (déplaçable) M_j et d'un ensemble de contraintes définissant géométriquement les placements stables pour les objets et les prises possibles.

Nous attacherons des symboles et des prédicats d'un langage de logique du premier ordre $\mathscr{L}$ à ces entités géométriques et à des propriétés spécifiques. La sémantique des symboles sera faite à travers quatre relations externes[2] : $\mathscr{R}_{robot}$, $\mathscr{R}_{obj}$, $\mathscr{R}_{space}$ et $\mathscr{R}_{property}$. Elles seront définies lors de leurs introductions dans ce chapitre.

Nous utilisons un langage PDDL2.1 de niveau 2 (avec typage et attribut numérique) [Long 03] pour notre représentation. Nous introduisons 5 symboles de type :

```
(:types robot obj subspace position property)
```

III.2.1 Des symboles pour les entités géométriques

Symboles des robots et des objets

Nous supposons que nous avons une représentation géométrique des robots et des objets. Cette représentation décrit :

– Un ensemble de polygones définissant les parties du robot ou des objets. C'est un modèle CAO.

– Les degrés de liberté existant entre les différentes partie d'un même robot. Ces degrés de liberté sont définit sous forme d'articulations entre les polygones décrivant un robot ou un objet. Ces articulations sont des joints (pivot, sphérique...) tels qu'ils peuvent être définis en Mécanique.

– Les méthodes cinématiques des robots.

Pour le problème interrupteur/radio, la représentation géométrique du robot `forklift` est $geo\_forklift$ et celle de l'objet `cbox`, *geo_cbox*. Nous utilisons deux relations

[2]externe car elles ne mettront pas en relation des éléments d'un même ensemble

$\mathscr{R}_{robot}$ et $\mathscr{R}_{obj}$ pour mettre en relation les symboles représentant les robots et les objets avec leurs pendants géométriques. Pour le problème interrupteur/radio cela nous donne :

$\mathscr{R}_{robot}(\texttt{forklift}, \textit{geo\_forklift})$ et $\mathscr{R}_{obj}(\texttt{cbox}, \textit{geo\_cbox})$ où `forklift` et `cbox` sont des symboles de type `robot` et `obj` respectivement. Les relations $\mathscr{R}_{robot}$ et $\mathscr{R}_{obj}$ mettent en relation des symboles de $\mathscr{L}$ à des entités géométriques R_i ou M_j de E.

Pour représenter les effets d'une action de prise, c'est à dire l'attachement d'un robot et d'un objet, nous considérons qu'une action de prise crée un nouveau robot. D'un point de vue géométrique, le nouveau robot est une composition du robot et de l'objet. Cette composition est définie comme les articulations pouvant exister dans un robot ou dans un objet seul.

Pour spécifier les compositions autorisées d'un point de vue symbolique nous introduisons le prédicat compose :

```
(compose ?r1 ?r2 - robot ?o - object)
```

Cela signifie que le robot `?r1` peut être obtenu en composant le robot `?r2` avec l'objet `?o`. Par exemple dans le problème interrupteur/radio, nous définissons :

```
(:objects
  forklift - robot
  cbox - obj
  forklift.cbox - robot
)

(compose forklift.cbox forklift cbox)
```

où le symbole de robot `forklift.cbox` est tel que

$\mathscr{R}_{robot}(\texttt{forklift.cbox}, \textit{geo\_forklift.geo\_cbox})$

Symboles des sous-espaces

Les différents sous-espaces définies de CS_{free} (les $CP(R_i)$, les $CP(M_i)$ et les $CG(R_i.M_j)$) sont représentés par des symboles de type `subspace`. La sémantique de ces symboles est donné à travers la relation $\mathscr{R}_{space}$. Par exemple, dans le problème interrupteur/radio, nous définissons les symboles de sous-espace suivants :

```
(:objects
  CP_forklift - subspace
  CP_armbot - subspace
  CP_cbox - subspace
  CG_forklift.cbox - subspace
)
```

avec :

$\mathscr{R}_{space}(\texttt{CP\_forklift}, CP(geo\_forklift))$

$\mathscr{R}_{space}(\texttt{CP\_armbot}, CP(geo\_armbot))$

$\mathscr{R}_{space}(\texttt{CP\_cbox}, CP(geo\_cbox))$

$\mathscr{R}_{space}(\texttt{CG\_forklift.cbox}, CP(geo\_forklift.geo\_cbox))$

$\mathscr{R}_{space}$ met en relation des symboles de $\mathscr{L}$ avec des ensembles de configurations appartenant à *CS*.

Symboles pour les sous-espaces spécifiques et pour les propriétés

Dans nos problèmes nous avons besoin de représenter des lieux spécifiques. En planification de tâches, il est fréquent d'utiliser des symboles pour dénoter des lieux. En général ces symboles ne sont pas reliés à une description précise du lieu qu'ils représentent. Par exemple dans ce cadre, pour le problèmes des tours de Hanoï, le fait qu'un cylindre soit sur une pile représenté par `(on_stack small-cylinder s1)` indique juste la présence d'une relation `on_stack` entre la pile `s1` et le petit cylindre `small-cylinder`. Aucune sémantique géométrique n'est donnée ici

Pour pallier à ce défaut, nous représentons les lieux par des symboles dit de *positions*. Les deux relations $\mathscr{R}_{space}$ et $\mathscr{R}_{property}$ vont nous permettre de donner une sémantique géométrique précise à ces symboles.

Nous distinguons trois types de positions :

– Les positions sans propriété particulière. Nous les appelons *positions à propriété de généralité*.

– Les positions où les robots peuvent appliquer des opérations de prise ou de pose des objets. Nous les nommons *position à propriété de lien*.

– Les positions où les robots sont initialement ou finalement ou les zones où les robots peuvent appliquer des opérateurs de haut niveau. Nous appelons opérateurs ou tâches de haut niveau les opérateurs ou les tâches ne représentant pas le mouvement

des robots ainsi que la prise ou la pose des objets. Nous nommons ces positions les *positions à propriété de zone*.

Nous définissons les positions comme appartenant à un des sous-espaces définis. Nous introduisons un prédicat pour représenter ceci :

```
(belongs-to ?p - position ?s - subspace)
```

Les positions à propriété de généralité. Pour une position de ce type, seul un fait `belongs-to` est nécessaire pour définir le lieu géométrique correspondant. Par exemple dans le problème interrupteur/radio, nous définissons la position où `forklift` n'est attaché à aucun objet avec :

```
(:objects
  forklift-anywhere - position
)

(belongs-to forklift-anywhere CP_forklift)
```

Le symbole de position `forklift-anywhere` représente n'importe quelle configuration appartenant à $CP(geo\_forklift)$.

Les positions à propriété de lien. Les sous-espaces où le système peut appliquer des opérations de prise ou de pose sont les $CP(R_i) \cap CG(R_i.M_j)$. Nous représentons ces sous-espaces à travers les symboles de position et du point de vue du robot seul, de l'objet seul et de la composition robot-objet. Par exemple dans le problème interrupteur/radio, nous définissons :

– `forklift-contact-cbox` représente les configurations de $CP(geo\_forklift)$ où *geo_forklift* est en contact avec *geo_cbox* suivant une prise autorisée par les contraintes de la manipulation.

– `forklift.cbox-stable` est la position représentant les configurations de $CG(geo\_forklift.geo\_c$ où *geo_cbox* est dans un placement stable tel qu'ils sont définis par les contraintes de la manipulation.

– `cbox-contact-forklift` est la position représentant les configurations de $CP(geo\_cbox)$ où `forklift` est en contact suivant une prise autorisée par les contraintes de la manipulation.

Ces différentes positions représentent des sous-espaces identiques. Seul le "point de vue" change : selon celui-ci nous aurons le robot et l'objet attaché ou pas. Cela nous permettra de définir clairement les opérations de prise et de pose. Nous représentons la similitude entre ces positions par le prédicat `link` :

```
(link ?p1 ?p2 - position)
```

Dans le problème interrupteur/radio, nous définissons :

```
(:objects
  forklift.cbox-stable - position
  cbox-contact-forklift - position
  forklift-contact-cbox - position
)

(link forklift.cbox-stable cbox-contact-forklift)
(link forklift.cbox-stable forklift-contact-cbox)
```

Les positions à propriété de zone. Pour ces positions, nous utilisons les symboles de propriété (type `property`). Au niveau symbolique ce symbole définit la propriété de haut niveau d'un lieu (i.e. d'une position) comme celle que le lieu est apte à la réception d'ondes radios. Au niveau sémantique, les symboles de propriété sont définis à travers la dernière relation externe : $\mathscr{R}_{property}$. Elle met en relation un symbole de propriété avec un ensemble de configurations. Dans la suite nous mettrons en relation le symbole de propriété avec une fonction délimitant cet ensemble de configurations.

Pour lier un symbole de position avec un symbole de propriété nous introduisons le prédicat `has-property` :

```
(has-property ?p - position ?pr - property)
```

Par exemple, toujours dans le problème interrupteur/radio, nous définissons :

```
(:objects
  forklift.cbox-ok-radio - position
  good-reception - property
)

(belongs-to forklift.cbox-ok-radio CG_forklift.cbox)
(has-property forklift.cbox-ok-radio good-reception)
```

Le symbole de position `forklift.cbox-ok-radio` représente n'importe quelle configuration de $CG(geo\_forklift.geo\_cbox)$ où `forklift.cbox` peut appliquer un opérateur de haut niveau `RECEIVE-DATA` (donné plus bas). Le symbole de propriété `good-reception` est lié à travers la relation $\mathscr{R}_{property}$ à une fonction spécifiant le sous-espace de configuration où la réception d'ondes radio est bonne. Nous rappelons que c'est seulement lorsque `forklift` transporte `cbox` que les données peuvent être reçues, ce qui explique pourquoi la position `forklift.cbox-ok-radio` est liée au sous-espace représenté par le symbole `CG_forklift.cbox`.

Nous pouvons également représenter les configurations initiales et finales des robots et des objets avec des positions à propriété de zone. Par exemple, nous définissons :

```
(:objects
  forklift-init - position
  init-forklift - property
)

(belongs-to forklift-init CP_forklift)
(has-property forklift-init init-forklift)
```

Le symbole de position `forklift-init` représente n'importe quelle configuration de $CP(geo\_forklift)$ où *geo_forklift* est sur une configuration initiale.

Nous avons représenté graphiquement sur la figure III.2 tous les symboles de positions définis pour le problème interrupteur/radio. Cette figure illustre comment les symboles de position sont liés entre eux à travers les prédicats `belongs-to` et `link`.

III.2.2 Topologie des espaces, topologie des états

Nous introduisons le prédicat `on` :

```
(on ?r - either obj robot ?p - position)
```

Ce prédicat nous permet de définir l'état complet du système. Par exemple l'état initial $s0$ est :

```
(and (on forklift forklift-init)
     (on armbot armbot-init))
     (on cbox cbox-init)))
```

Cela signifie que les robots et l'objet sont sur une de leurs configurations initiales. Dans nos problème, nous donnerons généralement qu'une seule configuration initiale pour chacun des robots et de objets de E. Si c'est le cas, cet état symbolique représente une configuration unique de CS. Généralement ce n'est pas le cas, par exemple ce nouvel état $s1$:

```
(and (on forklift.cbox anywhere-forklift.cbox)
     (on armbot armbot-switch))
```

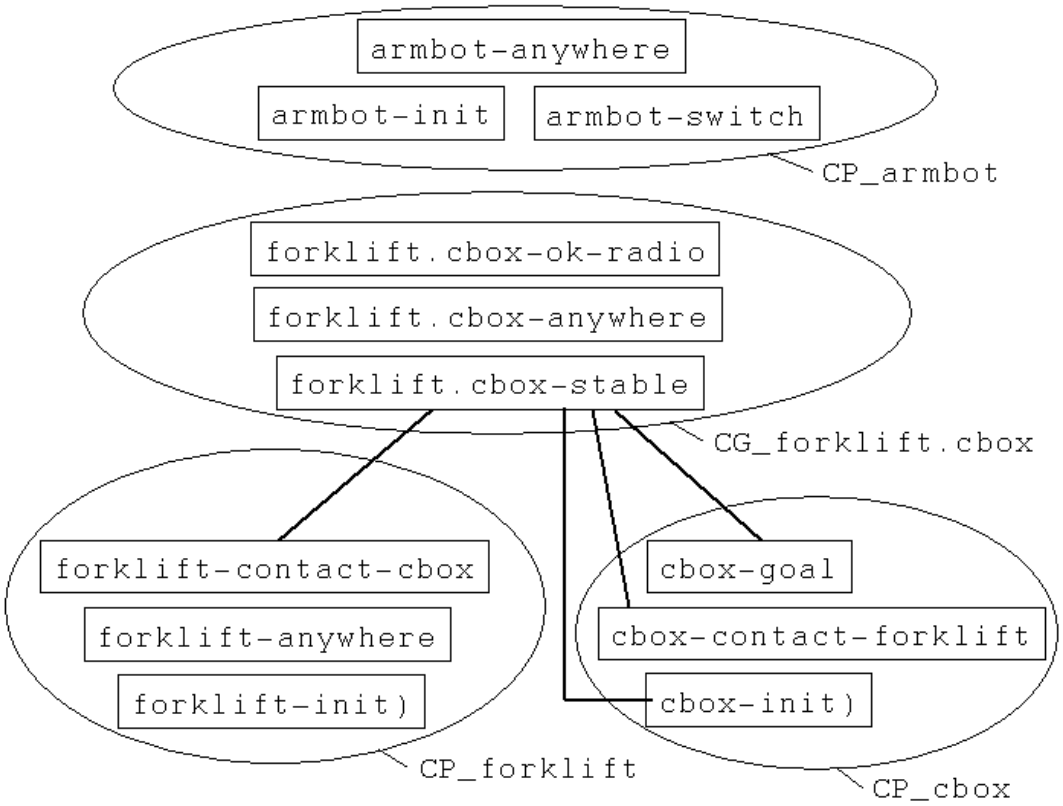

FIG. III.2 – Représentation graphique des prédicats `belongs-to` (les ellipses) et `link` (les lignes grasses) pour le problème interrupteur/radio.

Cet état signifie que `forklift` transporte `cbox` et que l'organe terminal de `armbot` est dans une position telle qu'il peut appuyer sur l'interrupteur. Cet état représente un ensemble infini de configuration : c'est l'ensemble défini par l'intersection des ensembles liés aux positions de l'état. L'état $s1$ correspond à l'ensemble $CG(geo\_forklift.geo\_cbox) \cap CP(geo\_armbot)$ réduit aux configurations décrites par la fonction liée au symbole de propriété `switch-push` lié au symbole de position `armbot-switch` (cette propriété décrit des configurations où le `armbot` est en position pour appuyer sur l'interrupteur). Nous verrons plus bas que nous appellerons cet ensemble CS_{s1}^{def}.

Chaque état est composé de quelques faits concernant le prédicat `on`. Nous voulons que le passage d'un état à un autre respecte les contraintes de la manipulation. Autrement dit, nous souhaitons que le passage d'un état à un autre corresponde à un chemin unique de transit ou de transfert ou d'une opération de prise ou de pose. Ce qui n'est pas le cas par exemple pour le passage de l'état $s0$ à $s1$. Pour cela, nous construisons un ensemble de trois opérateurs qui satisfasse cette propriété : `GOTO`, `GRASP` et `UNGRASP`.

L'opérateur GOTO signifie que le robot simple ou composé avec un objet peut se déplacer dans son sous-espace courant. Cela pourra correspondre à un chemin de transit ou de transfert.

```
(:action GOTO
 :parameters (?r - robot
              ?p1 ?p2 - position
              ?s - subspace)
 :precondition (and
                (on ?r ?p1)
                (belongs-to ?p1 ?s)
                (belongs-to ?p2 ?s)
 :effect       (and
                (not (on ?r ?p1))
                (on ?r ?p2))
)
```

GRASP est l'opérateur de composition d'un robot et d'un objet. UNGRASP est définie symétriquement.

```
(:action GRASP
 :parameters (?r - robot     ?p1 - position
              ?m - obj       ?p2 - position
              ?new-r - robot ?p3 - position)
 :precondition (and
                (on ?r ?p1)(on ?m ?p2)
                (compose ?r-new ?r ?o)
                (link ?p3 ?p1) (link ?p3 ?p2)
 :effect       (and
                (not (on ?r ?p1))
                (not (on ?r ?p2))
                (on ?r-new ?p3)))
```

La représentation basée sur une base de faits qui évolue au fil des actions appliquées est celle de la planification de tâches. Nous avons donc formé un problème et un domaine de planification de tâches. On peut remarquer que tous les prédicats définis, à part le prédicat on, sont statiques. Ils n'évoluent jamais dans les bases de fait représentant les états. Ces prédicats représentent grossièrement la topologie de l'espace des configurations (les composantes connexes des différents sous-espaces ne sont pas représentées à ce niveau) et les propriétés de certains des sous-espaces de *CS*. On pourra raisonner sur la représentation obtenue avec un planificateur de tâches classique.

III.2.3 Génération automatique des symboles

Nous avons présenté une méthode générale pour représenter les entités de E. Nous avons illustré cette méthode sur l'exemple du problème interrupteur/radio. Selon les problèmes, la conception de la représentation symbolique peut être fastidieuse. Par exemple, dans la version implémentée de la représentation symbolique pour le problème interrupteur/radio, nous introduisons une vingtaine de symboles et une cinquantaine de faits dans l'état initial. Nous avons alors en grande partie automatisé la production de symboles et de faits nécessaires.

En effet, avec peu d'entrées, nous sommes capables de générer automatiquement tous les symboles nécessaires. Avec la connaissance des robots disponibles, les objets, des compositions robots/objets et des zones à propriété (comme celle d'être une bonne zone de réception d'ondes radio), nous pouvons produire les symboles et leurs liens à travers les prédicats statiques définis. Par exemple, dans un problème interrupteur/radio réduit où nous ne considérons plus le robot `armbot` et où nous supposons que le flux de données radio existe en continu, nous spécifierons le fichier d'entrée suivant :

```
ROBOT:{FORKLIFT}
OBJECT:{CBOX}
COMPOSITION:{F_CBOX FORKLIFT CBOX}
SUBSPACE:{FORKLIFT CP ;
         CBOX CP ;
         F_CBOX CG}
LINK:{FORKLIFT CP / F_CBOX CG ;
      CBOX CP / F_CBOX CG}
ZONE:{F_CBOX CG GOOD_RECEPTION}
INIT:{FORKLIFT CP ;
      CBOX CP}
GOAL:{CBOX CP}
```

Notre générateur automatique de fichier PDDL2.1 produira l'état initial et l'état final. Il crée les symboles des 5 types définis nécessaires. Les symboles de position sont en général les plus nombreux. Nous avons donc adopté une nomenclature pour créer et pour faciliter les lectures des symboles de position. Cette nomenclature est :

```
P_{nom robot}_{nom sous-espace}_{propriété}
```

Chaque position est attachée à un sous-espace d'un robot. Pour la propriété, dans le cas des positions à propriété de généralité, la chaîne de caractères correspondante sera vide. Dans le problème interrupteur/radio réduit, le générateur produira :

– P_FORKLIFT_CP représente n'importe quelle configuration où forklift n'est attaché à aucun objet.

– P_F_CBOX_CG représente n'importe quelle configuration où forklift saisi cbox.

Dans le cas des positions à propriété de zone, la chaîne de caractère correspondante sera le nom de la propriété de la zone : INIT, GOAL ou du nom de la propriété (comme GOOD_RECEPTION par exemple). Dans notre exemple courant, nous aurons :

– P_FORKLIFT_CP_INIT la position représentant la configuration initiale de forklift.

– P_CBOX_CP_INIT et P_CBOX_CP_GOAL les positions représentant les configurations initiales de l'objet cbox.

– P_F_CBOX_CG_GOOD_RECEPTION la position représentant les configurations où le robot composé f_cbox est dans une zone de bonne réception.

Pour les positions faisant référence à des configurations où le système peut appliquer des opérations de composition ou de décomposition de robots et d'objets, le champ "propriété" fera explicitement référence au nouveau sous-espace auquel la position est liée via le prédicat link . Toujours dans le même exemple réduit du problème interrupteur/radio le générateur automatique de symboles produira :

– P_FORKLIFT_CP_CG_F_CBOX la position représentant les configurations. où forklift peut se composer avec cbox

– P_CBOX_CP_CG_F_C_BOX la positions représentant les configurations de cbox telles que l'objet peut être saisie par forklift

– P_F_CBOX_CG_CP_CBOX la position représentant les configurations de la composition f_cbox telles que l'objet cbox peut être déposé dans un placement stable.

Dans la suite de cette thèse nous garderons cette nomenclature pour les symboles de position.

III.3 Représentation finale des problèmes : un espace d'état "étendu"

Contrairement à une approche de planification de tâches classique, les symboles représentant les lieux (i.e. les symboles de position) sont maintenant liés à des espaces représentant la géométrie. Nous allons donc pouvoir lier le monde géométrique à un monde constitué de relations symboliques. Par la suite, nous formaliserons clairement l'espace d'état final obtenu. Pour cela nous allons étendre la représentation classique de la planification de tâches pour prendre en compte les préconditions géométriques pour l'application d'une action et les conséquences géométriques d'une telle application.

III.3.1 Combiner des relations symboliques

Les entités géométriques maintenant symboliquement représentées, nous pouvons leurs adjoindre des symboles non liés à la géométrie. Pour intégrer ces relations symboliques, nous nous appuyons sur un langage de logique du premier ordre $\mathscr{L}$ qui inclut les symboles déjà définis. Avec $\mathscr{L}$ nous pouvons construire de nouveaux opérateurs. Par exemple dans le problème interrupteur/radio, nous définissons un opérateur pour recevoir les données par ondes radio :

```
(:action RECEIVE-DATA
 :parameters (?r - robot
              ?p - position)
 :precondition (and
                (on ?r ?p)
                (has_property ?p good-reception)
 :effect       (data-received)
)
```

C'est typiquement une action qui possède une contrainte géométrique de placement exprimée par le prédicat `has_property`. Il est également possible de définir des opérateurs sans aucun lien avec le monde géométrique. Il est également possible de surcharger les opérateurs de base `GOTO`, `GRASP` et `UNGRASP`. Par exemple nous pourrions ajouter un fait `motor-lift-ok`, signifiant que les moteurs de l'élévateur sont en route, dans les préconditions de l'opérateur `GRASP`. Une action `MOTOR-ACTIVATION` devra être appelé pour mettre le fait `motor-lift-ok` à vrai. Cette action serait sans aucun lien avec la géométrie.

III.3.2 Représentation des domaines et des problèmes

Les définitions des domaines et des problèmes sont une extension de la représentation classique de la planification de tâches. Nous avons étendu les notations utilisées notamment dans [Ghallab 04].

Le domaine

Le domaine de planification que nous considérons est un système de transition état sur $\mathscr{L}$ s'appuyant sur CS et sur les quatre relations externes $\mathscr{R}_{robot}$, $\mathscr{R}_{obj}$, $\mathscr{R}_{space}$ et $\mathscr{R}_{property}$. Nous nommons le système de transition état $\Sigma = (S, A, \gamma)$.

Un état aura pour nous deux parties, la première sera un état symbolique et la seconde un sous-espace de CS. Ces deux parties sont indispensable à la définition d'un état dans notre cadre.

Soit S l'ensemble de ces états. S est inclut dans le produit cartésien de tous les états symbolique possible et de l'ensemble des partitions de l'espace des configurations CS.

Un état $s \in S$ est défini comme $s = (sym_s, CS_s)$ où :

– sym_s est un état symbolique tel qu'il peut être défini dans la représentation classique de la planification de tâches. Autrement dit sym_s est une conjonction de faits positifs. Nous nous plaçons dans l'Hypothèse du Monde Clos, tout ce qui n'est pas défini dans sym_s est considéré comme faux. La conjonction de faits de l'état symbolique sym_s peut contenir des faits mettant en jeu le prédicat "`on`". Nous rappelons que les faits "`on`" peuvent être supprimés ou ajoutés lors de la transition d'un état symbolique à un autre. La présence d'un fait `on` indique que le robot ou l'objet concerné est *actif* et donne sa *position* courante.

– CS_s est la partie géométrique de s. C'est un sous-espace de CS. Nous définissons CS_s^{def} comme étant l'intersection des sous-espaces étiquetés par les positions des robots et des objets actifs tel que nous les avons spécifiés dans sym_s. D'autre part, nous définissons CS_s^{reach} le sous-espace accessible de n'importe quelle configuration de CS_s. L'accessibilité désigne ici toutes les configurations atteignables par une seule étape de mouvement (de transit ou de transfert) par un des robots actifs.

Soit A l'ensemble des actions, c'est à dire l'ensemble de toutes les instances possibles des opérateurs tel qu'il pourrait être défini dans la représentation classique de la planification de tâches. Maintenant, nous voulons définir l'applicabilité d'une action a en un état s. Nous définissons également la fonction de transition γ qui associe un état s, une action a et un nouvel état s' si l'action est applicable. Nous rappelons qu'une action a est un triplet $\{name(a), precond(a), effects(a)\}$ où :

– $precond(a)$ est une conjonction de faits positifs $precond^+(a)$ et de faits négatifs $precond^-(a)$.

– $effects(a)$ est une conjonction de faits positifs $effects^+(a)$ et de faits négatifs $effects^-(a)$.

Nous définissons trois conditions pour l'applicabilité d'une action a sur un état s :

(p1) $precond^+(a) \subset sym_s$

(p2) $precond^-(a) \cap sym_s = \emptyset$

(p3) $CS_s \neq \emptyset$ (Si c'est la cas nous dirons que s est valide)

L'application d'une action a en état s créé un nouvel état s' à travers la fonction γ. Nous avons $\gamma(s,a) = s'$ avec :

(e1) $sym_{s'} = (sym_s - effects^-(a)) \cup effect^+(a)$

(e2) $CS_{s'} = CS_s^{reach} \cap CS_{s'}^{def}$

On peut observer que (p1), (p2) et (e1) correspondent à la définition classique des domaines de planification de tâches. (p3) est la condition qui assure que l'état a une réalité géométrique. On dit alors que l'état est valide. (e2) provoque la création de l'ensemble des configurations attachées à s'. Cet ensemble doit appartenir à $CS_{s'}^{def}$ par construction et doit être atteignable à partir de l'état s dans le respect des contraintes de la manipulation. Nous désirons faire deux remarques :

– $CS_s^{reach} \cap CS_{s'}^{def} \neq \emptyset$ aurait pu être une condition d'application de l'action a. Nous n'avons pas fait ce choix pour rester proche de l'implémentation qui suivra. Dans notre représentation, une action peut être applicable sans avoir de réalité géométrique, elle produira alors un état non valide sur lequel on ne pourra pas appliquer d'autres actions. Une solution au problème ne sera atteinte que si un état but valide est atteint (voir plus bas).

– Dans le cas d'action dite "purement symbolique", c'est à dire sans effets géométriques (i.e. sans ajout ou retrait de prédicats on dans ses effets), nous avons $CS_{s'} = CS_s$. Cela provient du fait que dans ce cas $CS_s^{def} = CS_{s'}^{def}$.

Le problème

Notre problème de planification est un triplet $P = (\Sigma, s0, g)$ où :

– $s0$ est l'état initial. En général nous choisirons $CS_{s0} = c_0$ un ensemble de configuration ne possédant qu'une seule configuration. Autrement dit, la configuration initiale de chaque robot et de chaque objet actif de $s0$ est unique.

– g est une conjonction de faits.

– Le but est spécifié comme $S_g = \{s \in S / sym_s\ satisfait\ g\ and\ CS_s \neq \emptyset (s\ est\ valide)\}$.

Une solution est trouvée si un planificateur atteint n'importe quel état appartenant à S_g.

Les définitions que nous proposons peuvent être vues indifféremment comme une extension de la planification de mouvement classique, comme une extension de la planification de manipulation multi-robots ou comme une extension de la planification de tâches classique. En effet on pourrait aisément exprimer :

– Un problème de planification de mouvement, si nous avons un seul robot et aucun objet déplaçable.

– Un problème de manipulation multi-robots, si nous ne considérons que les opérateurs `GOTO`, `GRASP` et `UNGRASP`.

– Un problème de planification de tâches classique si aucune action n'a de conséquences géométriques. C'est à dire qu'aucune action n'ajoute ou ne retire des faits mettant

en jeu le prédicat “on”.

Explorer un tel espace

Dans les problèmes réalistes, la construction de la topologie exacte de CS_{free} paraît hors d’atteinte. Par contre extraire des plans sans tenir compte de (p3) et de (e2) est un problème qui peut être considéré comme bien maîtrisé : c’est le problème de la planification de tâches classique. Un tel plan est appelé *plan possible*. Si nous construisons un plan possible (respectant (p1), (p2) et (e1)) et si nous trouvons au moins une configuration appartenant à CS_s pour chacun des états symboliques de ce plan, alors une solution est atteinte. En d’autres termes un *plan possible* dont on prouve que chaque état est valide est un plan solution.

Cette solution produira non seulement un plan respectant les contraintes relationnelles de haut niveau mais aussi les contraintes de la manipulation. La solution induira un chemin solution au problème de manipulation multi-robots sous-jacent. Autre remarque importante, la représentation symbolique est un problème relaxé où les interactions géométriques de bas niveau sont ignorées. On ne prend en compte dans la représentation symbolique que la topologie a priori de l’espace des configurations induite par les contraintes de la manipulation. La longueur d’un plan symbolique optimal en nombre d’action est donc une heuristique admissible pour nos problèmes. Nous verrons que l’utilisation des plans possibles est l’idée directrice de l’algorithme implémenté présenté plus loin.

Le problème que nous avons désormais à résoudre et de trouver les configurations qui nous permettrons d’atteindre une solution. La plupart de la complexité des problèmes est cachés dans les CS_s^{reach}. La construction exacte de la topologie ne semblant pas être atteignable, nous ferons appel à des approches probabilistes. C’est le sujet du prochain chapitre.

Chapitre IV

Utilisation des méthodes des roadmaps probabilistes

Dans le chapitre précédent, nous avons défini un lien entre un monde continu géométrique et un monde symbolique. Pour cela nous avons donné un “sens” aux opérations symboliques de placement et de déplacement des robots et des objets portés par les robots en les liant à des ensembles continus de configurations. Pour cela, nous avons définis un certain nombre de type de symbole et un jeu de prédicats spécifiques. Sur ces types et prédicats, nous avons construit des opérateurs (`GOTO`, `GRASP` et `UNGRASP`) définissant les mouvements et les manipulations et nous donnons la possibilité d'en créer d'autre ou de modifier ceux-ci. Une action construite avec des ajouts ou des retraits du prédicat spécifiant le placement (i.e. le prédicat `on`) fait référence à une multitude de problème de planification ou de manipulation.

Résoudre ces problèmes revient à caractériser l'espace des configurations *CS* de manière à trouver des configurations susceptible de valider les états. Malheureusement, dans les problèmes réalistes, c'est à dire avec un nombre important de degré de liberté, on ne peut pas connaître la topologie exacte de *CS*. Nous faisons alors appel à la méthodes des roadmaps probabilistes (PRM, cf. II.2.1) pour obtenir une caractérisation approchée de la topologie des ensembles continus de configurations. Chaque sous-espace induit par les contraintes de la manipulation (cf. III.1) sera représenté par une roadmap différente.

Dans ce chapitre nous montrons premièrement comment nous utilisons la méthodes des roadmaps pour représenter les différents sous-espaces et leurs intersections. C'est une extension des PRM pour la résolution des problèmes de manipulation multi-robots développée au LAAS-CNRS. La deuxième partie traite de méthodes avancées essentielles à la résolution de nos problèmes. Enfin, nous décrivons comment nous pouvons extraire une connaissance topologique globale à travers l'utilisation d'une multitude de roadmaps.

IV.1 Caractérisations des sous-espaces de configurations par différentes roadmaps

IV.1.1 Une roadmaps pour chaque sous-espace

Intrinsèquement, un problème de planification de mouvements est de complexité exponentielle en temps selon le nombre de degré de liberté pris en compte. Dans nos problèmes, le nombre de degré de liberté à considérer est la somme des degrés de liberté des robots et des objets.

Pour limiter le nombre de degré de liberté à considérer, nous choisissons de ne pas tenter dans un premier temps de caractériser la topologie de l'espace des configurations *CS* du système entier. En nous appuyant sur les définitions provenant des contraintes de la manipulation, nous savons qu'un chemin solution pour un problème de manipulation multi-robots est composé de chemins de transit et de chemin de transfert. Or ces chemins appartiennent à des sous-espaces précis de *CS*.

Nous allons alors construire une roadmap pour caractériser chacun des sous-espaces mis en jeu. Plus formellement, nous construisons une roadmap :

– pour chaque sous-espace $CP(R_i)(0 < i \leq r)$. Nous appelons ces roadmaps, *roadmaps de transit*, car c'est dans ces roadmaps que seront calculés les chemins de transit.

– pour chaque sous-espace $CP(M_i)(0 < i \leq m)$. Ces roadmaps sont appelés *roadmaps de placement*, car elles caractérisent les placements possibles des objets. Ces roadmaps ne possèdent pas d'arêtes, en effet les objets ne peuvent pas se mouvoir sans être transporter par un robot.

– pour chaque sous-espace $CG(R_i.M_j)(0 < i \leq r)(0 < i \leq m)$. Elles sont nommées *roadmaps de transfert* car c'est dans ces roadmaps que seront calculés les chemins de transfert.

Nous verrons plus loin (cf. IV.2.1) que dans le cas où les prises d'objets par les robots seraient continues, nous construirons également les roadmaps caractérisant les sous-espaces $CP(R_i) \cap CG(R_i.M_j)$.

Ces roadmaps sont construites indépendamment des autres robots et objets mais en prenant en considération l'environnement géométrique statique. Ainsi un noeud de la roadmap de transit caractérisant $CP(R_1)$ représente l'ensemble des noeuds de CS où le robot R_1 est dans une configuration précise et où les autres robots et objets sont dans une configuration quelconque.

Pour clarifier le propos, nous gardons l'exemple du problème interrupteur/radio. Sur la figure IV.1, nous montrons les roadmaps développées pour ce problème.

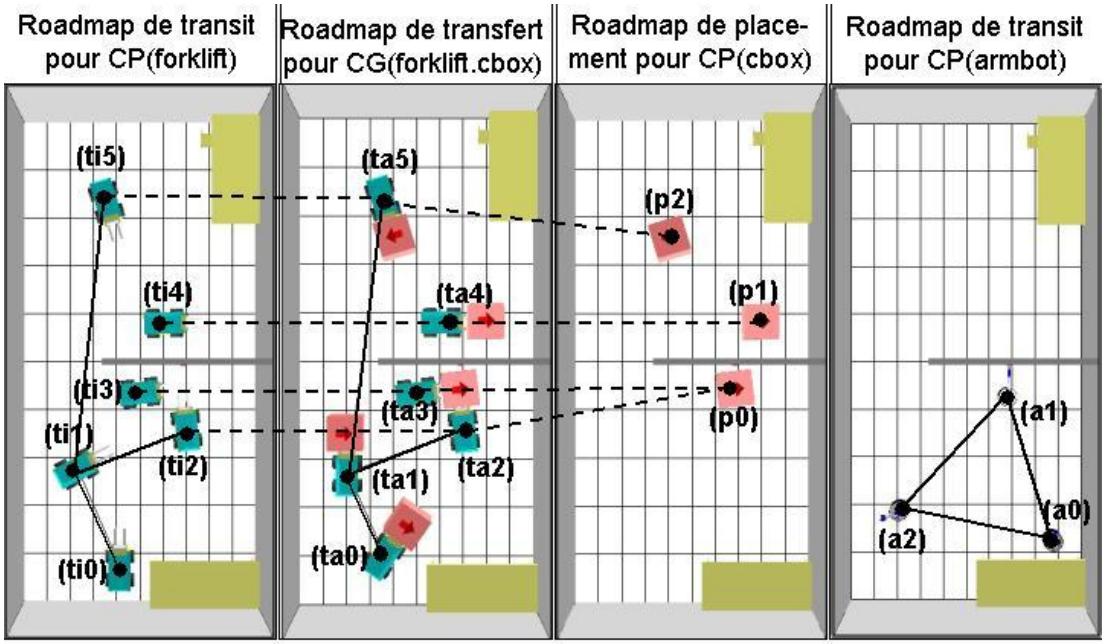

FIG. IV.1 – Roadmaps développées (lignes pleines) pour le problème interrupteur/radio. Les lignes discontinues sont les liens entre les roadmaps.

Les **roadmaps de transit** sont construites comme des roadmaps probabilistes classiques en planification de mouvements. Lors de la construction de ces roadmaps, nous ne prenons pas en compte les configurations possibles des objets ou des autres robots. Suivant ces configurations possibles de l'objet, des noeuds de configurations ou des arêtes de la roadmap de transit peuvent alors être invalides.

Les **roadmaps de transfert** sont également construites comme des roadmaps probabilistes classiques mais avec un nouveau robot qui est la composition d'un robot et d'un objet. Un robot et un objet peuvent se composer selon une des *prises* autorisées qui sont des données d'entrées du problème.

Les prises sont spécifiées comme une configuration relative de l'objet par rapport

au robot. D'un point de vue de l'implémentation, il n'y a pas de différence entre les articulations d'un robot et la prise du robot sur un objet.

Si pour un robot et un objet, nous définissons un nombre limité de prise, nous parlerons de *prises discrètes*. Par exemple, pour les problème de type "chariots élévateurs" (cf. I.2.1), nous avons défini quatre prises discrètes d'un chariot sur une boîte. Pour le problème interrupteur/radio, nous avons procédé de même. Sur les figures illustrant ces problèmes le lecteur reconnaîtra la prise discrète employée en regardant l'orientation de la flèche sur les boîtes.

Deux configurations d'une composition d'un robot et d'un objet mettant en jeu deux prises différentes ne peuvent jamais être reliées par un chemin local : c'est une des contraintes de la manipulation. Par exemple, sur la figure IV.1 il n'y a aucune chance de relier `(ta4)` et `(ta5)` par un chemin. Plus nous définissons de prises, plus le nombre de composantes connexes distinctes qui vont apparaître dans les roadmaps de transfert vont augmenter.

Nous pouvons également définir des prises comme un ensemble continu de configuration relative d'un objet par rapport à un robot. Par exemple, dans le problème IKEA (cf. I.2.4), les robots peuvent saisir les pieds de table en n'importe quel point. Ces prises, dite *continues*, sont définies comme les degrés de libertés du robot. Dans ce cas, il n'a y pas de limite quant au nombre de composantes connexes pour une roadmap de transfert. Si nous tirions complètement aléatoirement les configurations de la composition du robot et de l'objet, nous ne verrons probablement jamais apparaître d'arêtes dans la roadmap : chaque prise serait différente.

Pour pallier à ce problème, nous ne tirerons pas aléatoirement la prise à chaque tirage d'une nouvelle configuration. Nous stockerons les prises comme des prises discrètes et cette réserve de prises pourra être augmentée régulièrement. La seule différence entre les prises discrètes et les prises continues est donc que dans le deuxième cas le nombre de prises à considérer peut croître au cours de l'expansion de la roadmap.

L'utilisation de prises continues peut nous permettre de trouver une prise adéquate pour des problèmes de manipulation contraint. Toutefois, il faut être prudent pour les tirages des nouvelles prises : chacune d'entre elles provoque la création d'un nouvel espace à caractériser.

Les **roadmaps de placement** caractérisent les emplacements stables des objets. Comme pour les prises continues, chaque ajout de noeuds dans ces roadmaps induit en quelque sorte un nouveau sous-espace pour les autres roadmaps et augmente la combinatoire des interactions à évaluer entre les objets et les robots. Il ne faut ajouter des noeuds dans ces roadmaps que lorsque cela paraît nécessaire.

Résoudre un problème de manipulation multi-robots revient alors à créer des roadmaps et d'y trouver une série de chemins de transit dans les roadmaps de transit et de chemins de transfert dans les roadmaps de transfert. Il faut trouver un bon équilibre

entre les tirages dans la roadmap de transit, dans celle de transfert, et dans les sous-espaces $CP(R_i) \cap CG(R_i.M_j)$ (détaillé par la suite). Nous verrons que cette question de choix sur les roadmaps à étendre sera résolue (presque) indépendamment du problème (cf. V.2.6). Si les chemins sont valides dans les contextes courants des placements des robots et des objets, une solution est atteinte.

IV.1.2 Lier les roadmaps

Pour résoudre un problème de planification multi-robots, il ne suffit pas de caractériser chacun des sous-espaces de *CS* induit par les contraintes de la manipulation. Il faut également caractériser les parties de ces sous-espaces dans lesquels le système peut appliquer des opérations de prise ou de pose. Ces sous-espaces sont les $CP(R_i) \cap CG(R_i.M_j)$. Ceci doit également apparaître au niveau des roadmaps.

Sur notre exemple, la roadmap de transfert représente le sous-espace $CG(forklift.cbox)$. Dans cette roadmap, il y a des configurations qui peuvent être décomposé en une configuration de $CP(forklift)$ et une configuration de $CP(cbox)$. Sur la figure IV.1, ces décompositions possibles sont représentés sous forme de lignes discontinues appelées *liens*. Ces liens peuvent être mis en correspondance avec le prédicat `link` (cf. III.2). En effet les positions liées par le prédicat `link` font référence à des configurations pouvant être liées par des liens de roadmap.

Si nous construisions toutes ces roadmaps de manière totalement indépendante les unes des autres, les chances de trouver des liens entre les configurations des différentes roadmaps sont quasi nulles. Nous utilisons donc une méthode de construction active de ces configurations liées. Nous pouvons procéder par calcul de cinématique directe ou de cinématique inverse.

Pour la méthode de cinématique directe, nous tirons une configuration de la composition robot/objet en forçant certains de ses degrés de liberté de manière à ce que l'objet soit en position stable. Nous déduisons ensuite les configurations du robot seul et de l'objet seul. Malheureusement, dans la plupart des problèmes de manipulation, les données d'entrée contiennent uniquement les configurations initiales et finales des objets. Il nous donc faut pouvoir trouver les configurations de la composition du robot et de l'objet correspondant à ces placements : c'est la méthode de cinématique inverse.

Pour la méthode de cinématique inverse, nous procédons ainsi :

– Choix d'une configuration de l'objet. On sélectionne ou on tire un noeud dans la roadmap de placement de l'objet.

– Sélection d'une prise ou tirage aléatoire d'une nouvelle prise.

– Calcul de cinématique inverse permettant de trouver une instanciation pour tous les degrés de liberté du robot.

La dernière étape a été grandement optimisée en remarquant que la base du ro-

bot doit rester sur le sol. Le système composé du sol de l'objet et du robot forme alors une boucle qui doit rester fermée. L'optimisation repose essentiellement sur la détermination de la zone dans laquelle doit se trouver la base du robot (i.e. le corps du robot qui est en contact avec le sol). Le lecteur intéressé trouvera de plus amples informations sur cet algorithme nommé RLG dans la thèse de Juan Cortés ([Cortés 03]) ou dans les articles [J.Cortés 02] et [J.Cortés 03]. Le RLG démontre toute son efficacité dans le cas des robots possédant des degrés de liberté redondants.

Une fois que nous avons trouvé une configuration de $CP(R_i) \cap CG(R_i.M_j)$, nous pouvons l'ajouter telle quelle dans la roadmap de transfert caractérisant $CG(R_i.M_j)$. Nous ajoutons la configuration du robot seul dans la roadmap de transit caractérisant $CP(R_i)$. Cette opération d'ajout automatique est appelée *héritage*. Un lien d'héritage permet de reproduire fidélement une partie de la configuration d'une roadmap dans une autre. Plus formellement un héritage est une projection d'une configuration dans un espace des configuration de dimension plus faible.

Les roadmaps de transfert et de transit sont alors dites liées par un *lien d'héritage*. Si pour cette configuration, nous avions de plus trouvé une nouvelle configuration de l'objet, nous l'ajouterons à la roadmap de placement. Si nous avions également fait usage d'une nouvelle prise dans le cas des prises continues, nous la stockerions dans la "réserve" de prise. Les derniers mécanismes décrit sont également des héritages.

IV.1.3 Utiliser les définitions des domaines avec les roadmaps

Avec cette méthode de roadmaps différentes pour chaque sous-espace, une configuration unique de *CS* est obtenue en choisissant une configuration (i.e. un noeud) dans chacune des roadmaps des robots actifs d'un état.

Dans nos problèmes nous choisirons souvent une configuration unique pour l'état initial $s0$ du problème. Dans le problème interrupteur/radio, la part symbolique de $s0$, sym_{s0} est (en utilisant la nomenclature proposée en III.2.3) :

```
(on FORKLIFT P_FORKLIFT_CP_INIT)
(on ARMBOT P_ARMBOT_CP_INIT)
(on CBOX P_CBOX_CP_INIT)
```

Via la relation externe $\mathscr{R}_{property}$, les positions finissant par `INIT` représentent n'importe quelles configurations de *CS* où leurs robots ou objets respectifs se trouvent sur leurs configurations initiales. La relation $\mathscr{R}_{property}$ nous donne donc la part géométrique de $s0$: CS_{s0} = (`(ti0)`,`(p0)`,`(a0)`) (voir figure IV.1).

Imaginons maintenant que nous voulons appliquer l'action

```
(GOTO FORKLIFT P_FORKLIFT_CP_INIT P_FORKLIFT_CP_CG_F_CBOX)
```

Cette action est applicable car elle respecte les conditions d'application (p1) (p2) et (p3) d'une action sur un état (cf. III.3.2). Elle produit un état $s1$ dont la partie symbolique sym_{s1} est égale à :

```
(on FORKLIFT P_FORKLIFT_CP_CG_F_CBOX)
(on ARMBOT P_ARMBOT_CP_INIT)
(on CBOX P_CBOX_CP_INIT)
```

CS_{s1}^{def} est l'intersection des sous-espaces définis par les positions de l'état. L'utilisation des roadmaps va nous définir un ensemble fini de configurations qui appartiennent à cette intersection. La position `P_FORKLIFT_CP_CG_F_CBOX` identifie toutes les configurations où `forklift` est en contact avec `cbox` de manière à former un nouveau robot `f_cbox`. De plus la position `P_CBOX_CP_INIT` indique que l'objet `cbox` est dans sa configuration initiale. A cela s'ajoute la position `P_ARMBOT_CP_INIT` qui indique que le robot `armbot` doit être sur sa configuration initiale `(a0)`.

Nous reconnaissons ces configurations au niveau des roadmaps grâce notamment aux liens : toutes les configurations de la roadmap de transit du robot `forklift` ayant un lien avec une configuration de la roadmap de transfert de la composition de robot `f_cbox` qui elle même est liée à la configuration initiale de `cbox (p0)` sont retenues. De plus `armbot` doit être sur sa configuration initiale `(a0)`.

Les configurations de CS_{s1}^{def} sont alors pour nous : `((ti2),(p0),(a0))` et `((ti3),(p0),(a0))`.

CS_{s0}^{reach} est l'espace accessible en une seule étape de mouvement respectant les contraintes de la manipulation à partir de n'importe quelle configuration appartenant à CS_{s0}. Au niveau des roadmaps, le système global peut avoir accès à la composante connexe du noeud `(ti0)` (4 noeuds) et à celle du noeud `(a0)` (3 noeuds), ce qui nous donne douze configurations pour CS_{s0}^{reach}.

Nous obtenons un seul candidat pour $CS_{s1} = CS_{s0}^{reach} \cap CS_{s1}^{def}$: `((ti2),(pl0),(a0))`. Ce n'est qu'un candidat car les roadmaps ont été construites sur la base de l'environnement statique, donc il n'est pas sûr que cette configuration soit effectivement atteignable sans collision à partir de la configuration initiale. Il est possible par exemple que le chemin local menant de `(ti0)` à `(ti2)` soit en collision avec le placement `(p0)`.

Il nous faudra donc une deuxième étape de calcul pour que ce candidat se transforme en un réel élément de CS_{s1}. Cette seconde étape sera appelé "validation" (cf. V.2.5). Si nous réussissons à "valider" cette configuration candidate, nous aurons $CS_{s1} \neq \emptyset$: $s1$ devient valide.

IV.1.4 Conclusion sur la méthodes des roadmaps hétérogènes

L'approche par roadmaps hétérogènes permet de casser la complexité intrinsèque des problèmes de planification de manipulation multi-robots lors de la construction

des roadmaps. Les sous-espaces étudiés sont au maximum de la dimension du robots possédant le plus de degré de liberté.

Comme nous venons de le préciser, cette méthode ne peut pas être complète sans une deuxième étape de calcul qui vérifiera les interactions géométriques entre les robots et les objets de E. Cette méthode peut être vue comme méthode de raffinage en deux étapes. La première étape de calcul, la construction des roadmaps, s'assurent du respect de la non-collision des robots avec l'environnement statique. La seconde étape s'assurent de la non collision des robots et des objets entre eux.

C'est une méthode générale à laquelle il faut adjoindre des méthodes particulière pour en améliorer les performances.

IV.2 Méthodes avancées

Nous avons présenté la base de notre approche basée sur l'utilisation de roadmaps hétérogènes. Nous décrivons ici deux méthodes avancées basées également sur les méthodes des roadmaps probabilistes servant à améliorer les performances de la méthode générale.

IV.2.1 Les prises continues et les manipulations complexes

Dans certains problèmes de manipulation complexes, où les prises sont continues, il est fréquent de devoir saisir plusieurs fois l'objet pour le dégager d'un obstacle afin de le transférer à la configuration finale souhaitée. Une caractérisation plus importante des espaces $CG(R_i.M_j) \cap CP(R_i)$ peut alors être très utile. Une roadmap est alors entièrement dévolue à ce rôle.

La roadmap de transit et la roadmap de placement ne seront plus liées directement à la roadmap de transfert caractérisant $CG(R_i.M_j)$. Ces roadmaps seront liées via des liens d'héritage avec une roadmap caractérisant $CG(R_i.M_j) \cap CP(R_i)$.

Ce type de roadmap que nous nommons roadmap de *grasp* $\cap$ *placement* est construite en autorisant la prise continue à évoluer le long des chemins locaux. Ces chemins ne correspondent pas à des mouvements autorisés par les contraintes de la manipulation mais possèdent une bonne propriété dite de *réduction* qui permettent de les convertir en une séquence finie de chemins de transit et chemins de transfert. Une démonstration de cette propriété peut être trouvée dans [Alami 95].

La figure IV.2 illustre l'utilisation de cette propriété sur un exemple de manipulation contraint. Un bras manipulateur doit dégager une barre d'une cage. La première vignette montre un mouvement calculé dans une roadmap de grasp $\cap$ placement. Lors de ce mouvement, seule la barre se déplace : la prise continue évolue pendant le mou-

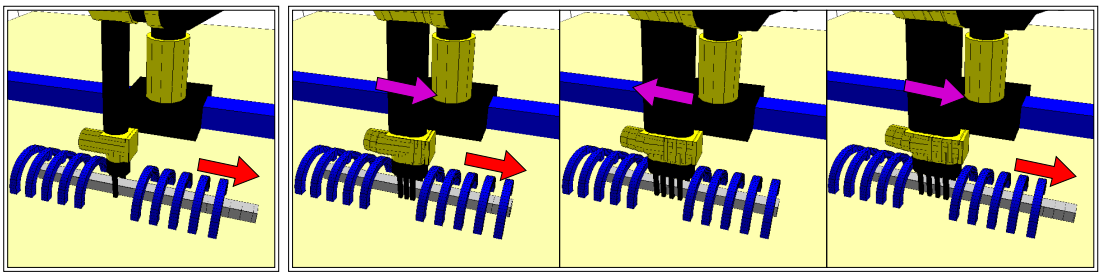

FIG. IV.2 – Illustration de l'utilisation de la propriété de réduction.

vement. Les trois autres vignettes montrent comment ce mouvement est décomposé en un mouvement de transfert, un mouvement de transit puis un mouvement de transfert.

Cette procédure nous permet de détecter plus rapidement les changements de prise "majeurs" à effectuer. En effet si on réussit à trouver un chemin entre le noeud de $CG(R_i.M_j) \cap CP(R_i)$ correspondant à la configuration initiale de l'objet et le noeud de $CG(R_i.M_j) \cap CP(R_i)$ correspondant à la configuration finale de l'objet au sein de la roadmap de grasp $\cap$ placement, nous pouvons alors extraire une solution sans changement majeur de prise : l'utilisation de la propriété de réduction suffira.

Si on ne trouve pas la composante connexe reliant la configuration initiale à la configuration finale de l'objet à travers la roadmap de grasp $\cap$ placement, c'est qu'il faut peut-être faire un changement majeur de prise. On construira alors des arêtes dans la roadmap de transit pour atteindre une nouvelle prise de l'objet. On peut supposer alors que la configuration initiale et la configuration finale ne peuvent être reliées qu'en visitant deux composantes connexes distinctes de $CG(R_i.M_j) \cap CP(R_i)$ dans la roadmap de grasp $\cap$ placement. La figure IV.3 montre un mouvement de transit permettant d'aller visiter une nouvelle composante connexe de la roadmap de grasp $\cap$ placement. Des arêtes dans la roadmap de transfert peuvent être également nécessaires pour trouver une solution.

Cette méthode est issue des travaux de thèse [Sahbani 03]. Originellement, une seule roadmap est développée : les arêtes de cette roadmaps peuvent alors représenter un chemin local de transit, de transfert ou de grasp $\cap$ placement. Nous avons adapté cette méthode pour la rendre compatible avec notre usage de plusieurs roadmaps hétérogènes.

IV.2.2 Le problème de l'approche de l'objet

Dans notre approche du problème de la manipulation multi-robots, nous construisons les roadmaps de transit indépendamment des objets. La résolution du sous problème de l'approche de l'objet devient alors difficile. En effet nous aurions peu de chance de

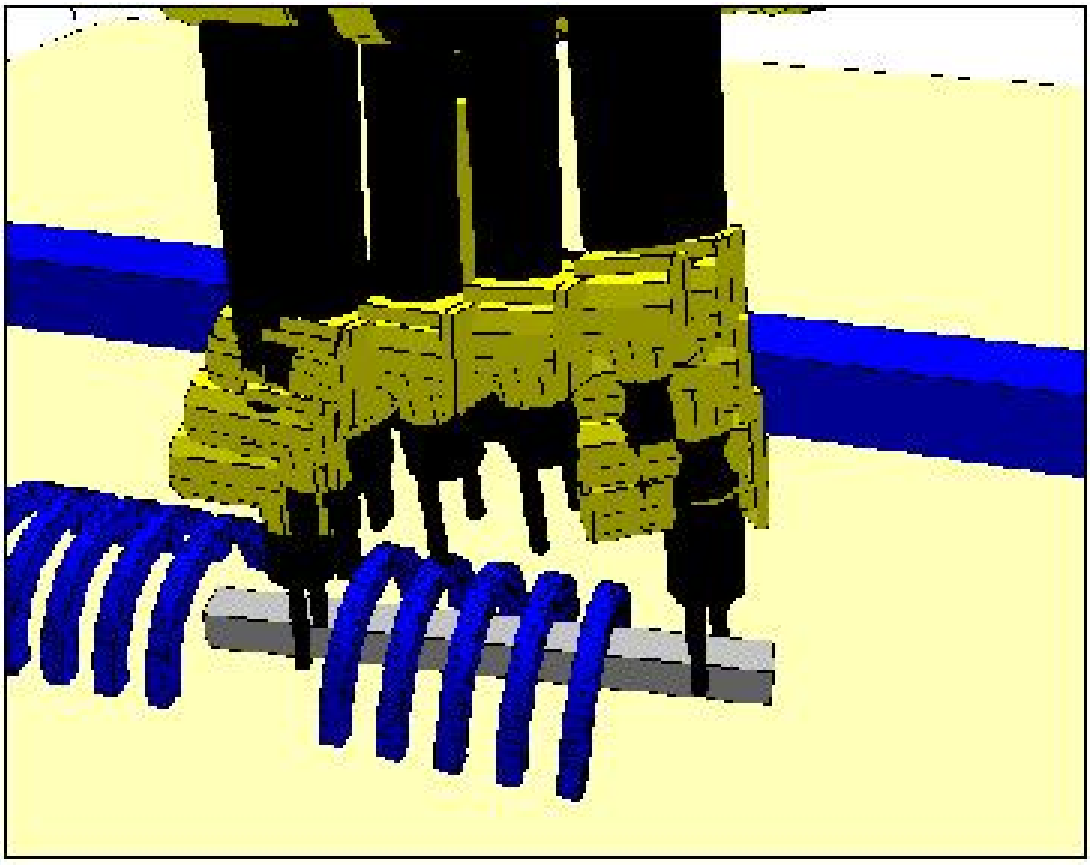

FIG. IV.3 – Un mouvement de transit est nécessaire pour aller visiter une nouvelle composante connexe de la roadmap de grasp ∩ placement.

trouver des chemins locaux sans collision avec l'objet vu que celui-ci n'est pas pris en compte. Nous avons vu précédemment comment obtenir la configuration "finale" où le robot est en position de prise avec l'objet (cf. IV.1.2), mais cela ne nous dit pas comment l'atteindre.

De plus, ce sous problème peut réapparaître plusieurs fois, si le robot doit changer de prise par exemple. Ce sous problème de l'approche de l'objet peut être également relativement compliqué, vu qu'il se ramène à un problème de planification de mouvement en environnement contraint. Pour toutes ces raisons, il nous a paru intéressant de développer une roadmap spécifique à la procédure d'approche que nous appelons *roadmap relative*. Cette roadmap va nous permettre de capitaliser les résultats obtenus. De plus, si la roadmap de transit peut trouver avantage à être construite par une méthode

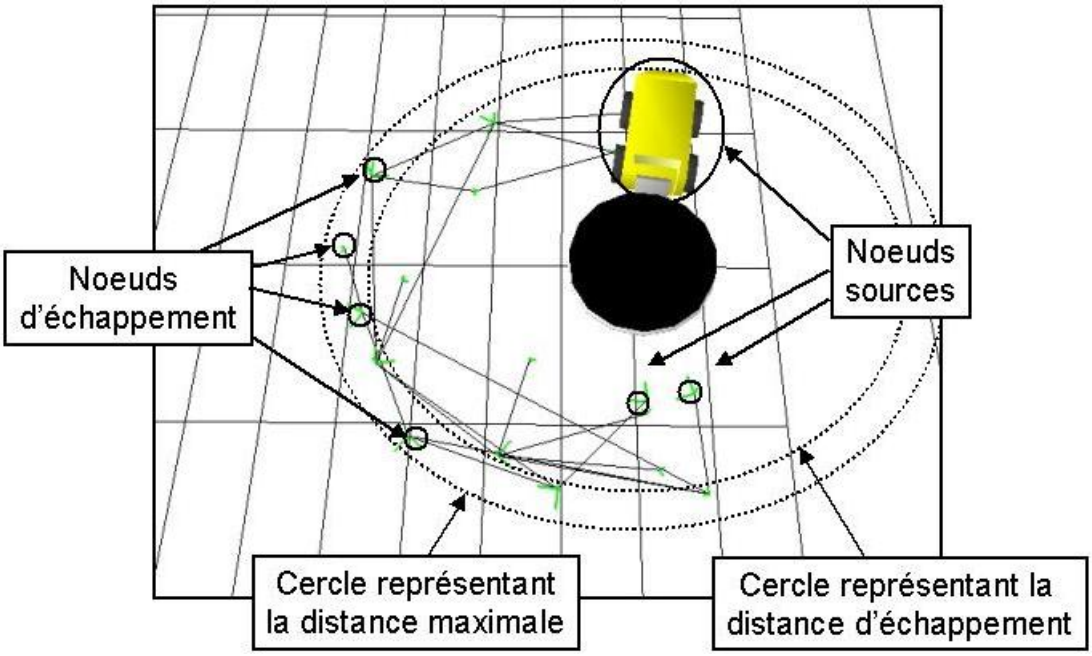

FIG. IV.4 – Une roadmap relative.

PRM, la roadmap relative pourrait avoir avantage à être développée par une méthode de type RRT connue pour avoir de meilleurs résultats sur les problèmes contraints.

Nous construisons les roadmaps relatives pour un robot et relativement à un objet et cela sans tenir compte de l'environnement statique. Nous lui donnons des fonctions de changement de repère pour qu'elle puisse être placée n'importe où dans l'environnement. Elle pourra ainsi s'adapter à n'importe quel placement de l'objet. Nous lui attribuons également deux distances. La première, la *distance maximale* définit la zone de tirage dans laquelle on va construire cette roadmap. La seconde, la *distance d'échappement* est inférieure à la distance maximale. Elle définit la distance à laquelle nous pensons que le robot est dégagé de l'objet. Ces deux distances sont données par le développeur du problème. Les noeuds compris entre ces deux distances sont appelés *noeuds d'échappement*. Tout ceci est illustré sur la figure IV.4 pour le sous problème de l'approche du gros cylindre dans le problème des tours de Hanoï (cf. I.2.3)).

L'utilisation des roadmaps relatives est illustrée sur la figure IV.5.

L'utilisation d'une roadmap relative met en jeu trois roadmaps pour un robot R_i et un objet M_j :

– La roadmap relative elle-même.

– la partie $CG(R_i.M_j) \cap CP(R_i)$ de la roadmap de transfert ou la roadmap grasp $\cap$

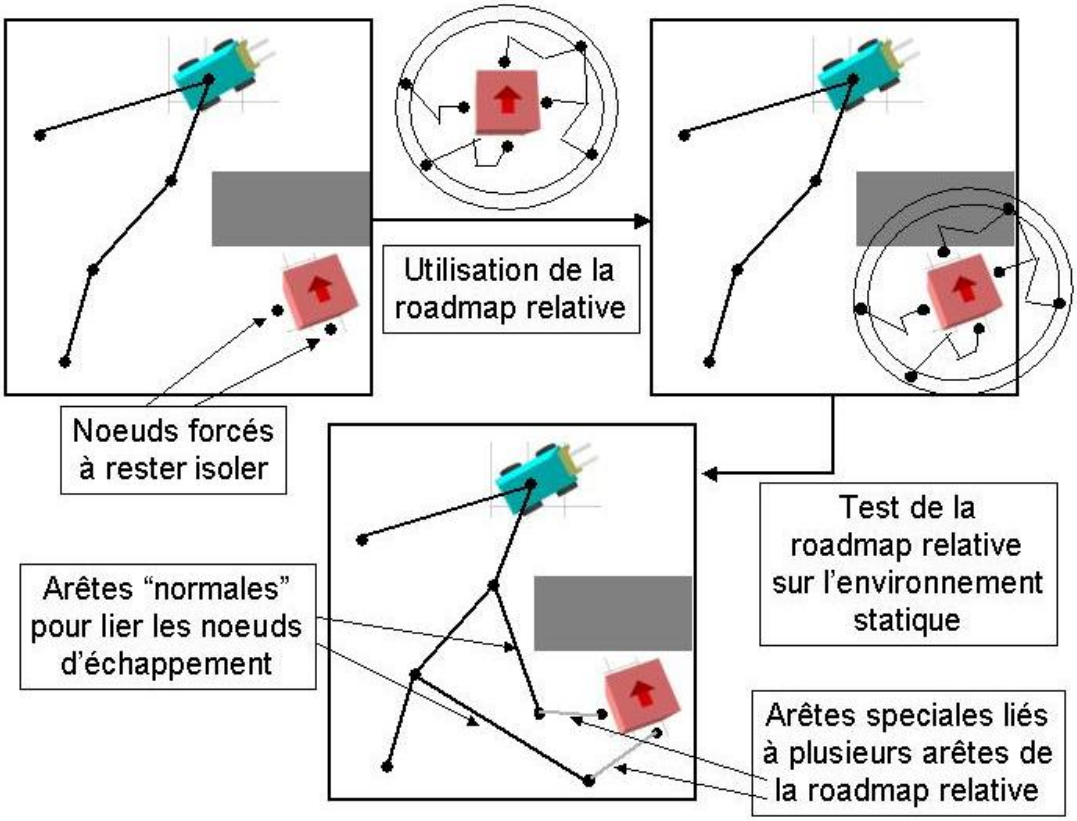

FIG. IV.5 – Utilisation d'une roadmap relative.

placement si on a jugé utile de développer une telle roadmap (cf. IV.2.1).

– la roadmap de transit.

Ces trois roadmaps seront liées de manière particulière. Chaque noeud calculé dans l'espace $CG(R_i.M_j) \cap CP(R_i)$ est hérité, après le changement de repère adéquat dans la roadmap relative. Ces noeuds de la roadmap relative sont appelés *noeuds sources*. La roadmap de transit, elle, héritera également de ces configurations des configurations du robot seul. On force ces noeuds ajoutés dans la roadmap de transit à rester isolés. Ils ne peuvent pas être reliés à d'autres noeuds de le roadmap de transit par les méthodes d'expansion classiques.

Une fois la roadmap relative suffisamment développée, les noeuds qu'elle aura hérités de l'espace $CG(R_i.M_j) \cap CP(R_i)$ seront reliés à des noeuds d'échappement. Les noeuds d'échappement seront alors reportés, après le changement de repère adéquat, dans la roadmap de transit. On vérifie alors les collisions entre les arêtes de la road-

map relative et l'environnement statique. Il est inutile par contre de vérifier les collisions avec l'objet : la construction de la roadmaps relative garanti la non-collision. S'il existe un chemin entre les noeuds sources et les noeuds d'échappement sans collision avec l'environnement, les noeuds de transit hérités des noeuds sources et des noeuds d'échappement de la roadmap relative sont reliés dans la roadmap de transit par une arête spéciale. Cette arête peut en fait représenter plusieurs arêtes de la roadmap relative. Autrement dit, on ne reporte dans la roadmap de transit que les noeuds sources et les noeuds d'échappement (après le changement de repère adéquat). Ce qui limite l'ajout de noeuds et d'arêtes dans la roadmap de transit.

Cette technique est très importante dans toute notre approche. Généralement, nous ne pourrons pas résoudre nos problèmes sans elle.

IV.3 Une multitude de roadmaps

L'approche que nous venons de détaillée reposant sur la construction de roadmaps hétérogènes est développée conjointement avec F. Gravot [Gravot 03a] et [Gravot 02]. Les roadmaps relatives sont un ajout à cette approche dont l'utilité ne nous est apparue que lors de la conception du planificateur aSyMov (cf. V).

Pour résumer, notre approche de résolution de problèmes de manipulation multi-robots se base sur :

– Chaque type de mouvement possible est représenté par une roadmap différente. Nous considérerons d'ailleurs souvent dans le reste de cette thèse qu'une roadmap de transfert n'est qu'une roadmap de transit mettant en jeu un "nouveau" robot, composition du robot seul et de l'objet qu'il transporte.

– Les roadmaps sont construites indépendamment des autres robots et objets. Le nombre de degré de liberté à considérer reste ainsi limité.

– On peut s'aider de roadmaps *heuristiques* tel que les roadmaps relatives et les roadmaps de grasp $\cap$ placement. Nous nommons ces roadmaps *heuristiques* car elles ne correspondent pas à des mouvements admissibles mais elles aident à la recherche de chemins solutions.

Cette approche présente un inconvénient majeur. En planification de mouvement classique, le fait qu'une configuration q_i et q_f soient dans une même composante connexe implique qu'il existe un chemin entre q_i et q_f. Avec notre approche, l'appartenance pour deux configurations à une composante connexe n'implique pas forcément l'existence d'un chemin. Cela implique plutôt que ce chemin existe pour un ensemble de configurations des autres robots et objets de l'environnement. Nous appellerons cette connexité *connexité faible*. C'est bien une notion *faible* dans le sens où elle est une condition nécessaire à la découverte d'un chemin et non pas une condition suffi-

sante.

Il nous a tout de même semblé intéressant d'étendre cette notion de connexité faible à travers la multitude des roadmaps. C'est à dire de trouver une condition nécessaire à l'existence d'une solution à un problème de manipulation multi-robots au sein de la multitude des roadmaps. Les *Graphes de Connexités Élémentaires* ou *G.C.E* permettent de dégager cette notion. Ils sont expliqués ci-après.

IV.3.1 Représenter la multitude de roadmaps

Même pour un problème simple en apparence tel que celui des chariots élévateurs (I.2.1), le nombre de roadmaps à utiliser peut être relativement important. En effet si nous considérons un problème de type chariots élévateurs plus réalistes avec :

– deux robots et une boîte.

– les prises de la boîte par les chariots sont continues.

– les chariots peuvent se passer directement la boîtes dans une position non stable pour la boîte. C'est à dire que les deux robots peuvent se passer la boîte sans la déposer.

Le nombre de roadmap devient relativement important. Il nous faudrait :

– Une roadmap pour caractériser les placements stables de la boîte.

– Deux roadmaps de transit pour les deux chariots se mouvant seuls dans l'environnement.

– Deux roadmaps de grasp∩placement avec relaxation de la prise continues. Il nous faut une roadmap pour chaque composition robot-objet. Or, nous avons deux robots et un objet.

– Deux roadmaps de transfert.

– Une roadmap de transfert pour la composition des deux robots avec la boîte. Cette roadmap est spécialisée dans le passage de la boîte entre les deux robots quand celle-ci n'est pas en position stable (comme par delà la fenêtre).

Ce problème est illustré sur la figure IV.6. Nous avons également représenté sur cette figure l'ensemble des roadmaps

Cette figure a pour but de montrer la complexité de notre "espace" de roadmaps. Le besoin d'un outil devient nécessaire pour savoir si l'état final du problème est possiblement atteignable à partir de l'état initial. Pour cela nous appliquons une transformation sur notre structure de roadmap pour obtenir ce que nous avons appelé les Graphes de Connexités Élémentaires.

Sur la figure IV.7 nous montrons une illustration simplifiée de cette transformation pour l'exemple décrit sur la figure III.1. Cet exemple est celui d'une boîte à transférer d'une configuration à une autre. Le transfert n'est possible qu'en déposant la boîte sur une configuration intermédiaire (`p2`). Sur cette figure, nous pouvons remarquer qu'il

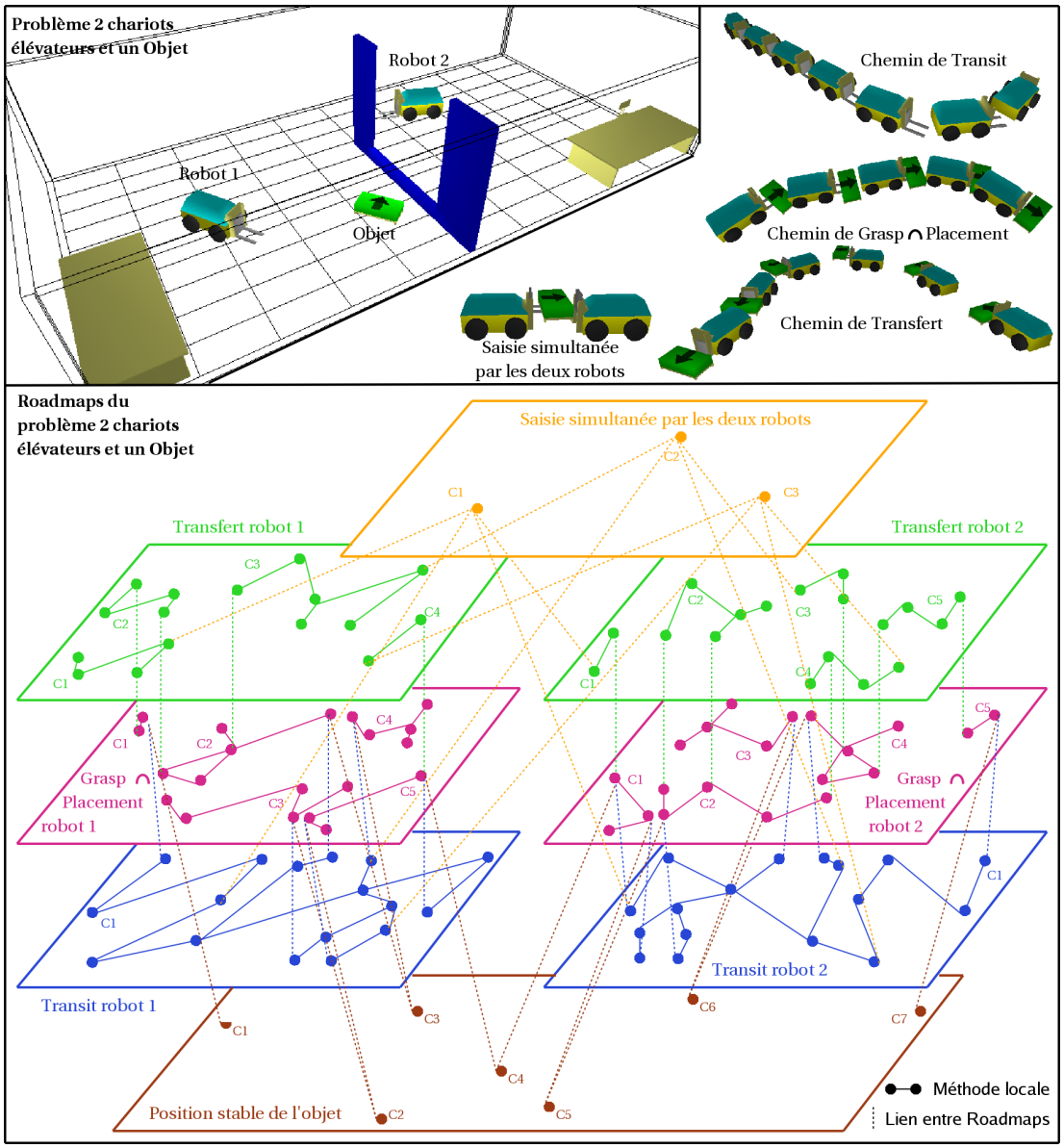

FIG. IV.6 – Les roadmaps, leurs liens dans un problème des chariots élévateurs

n'y a qu'une seule composante connexe pour la roadmap de transit alors que celle-ci représente trois sous-espaces distincts (un par placement de l'objet). Cela est dû à la construction de la roadmap de transit sans prendre en compte les configurations de l'objet. La roadmap de transfert, par contre, présente deux composantes connexes : une

par prise.

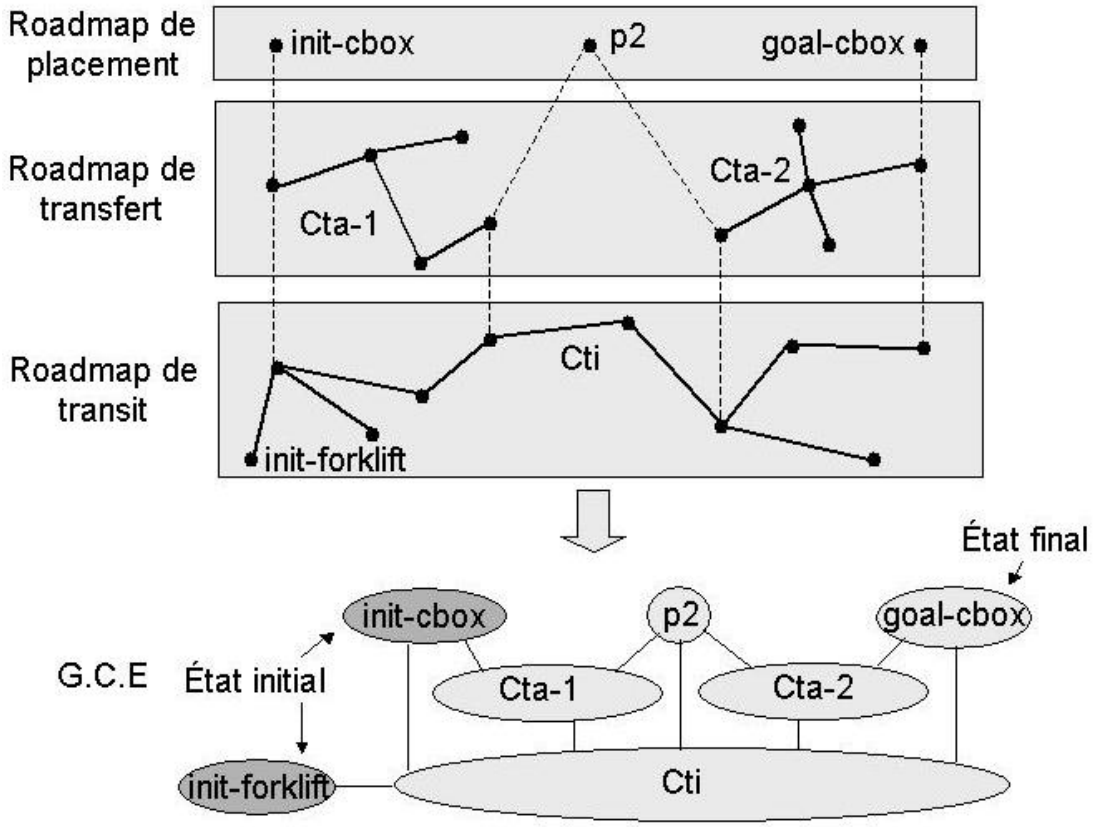

FIG. IV.7 – Exemple de G.C.E.

Cette nouvelle structure a pour but de mettre en relation les composantes connexes des différentes roadmaps. L'algorithme de création d'un G.C.E. crée un noeud dans un graphe pour chaque composante connexe de chaque roadmap. Il relie les noeuds du G.C.E entre eux suivant que la composante connexe associée est liée par au moins un noeud à une composante connexe d'un autre graphe. Lors des expansions des roadmaps, le G.C.E est mis à jour. Si deux composantes d'une roadmap s'unifient, ce qui arrive lorsqu'un nouveau noeud peut être lié via une arêtes à deux composantes connexes jusque là distinctes, des noeuds du G.C.E vont s'unifier également pour n'en former plus qu'un.

L'état initial et l'état final de nos problèmes correspondent à plusieurs noeuds dans diverses roadmaps. Avec une telle structure, on peut savoir si l'état initial est d'une certaine manière connexe à l'état final. Nous appelons cette "connexité" la *G.C.E-*

connexité. L'algorithme de vérification de G.C.E-connexité localise les noeuds initiaux dans le G.C.E puis diffuse ceux-ci dans les autres noeuds du G.C.E. La diffusion est terminée lorsqu'on ne peut plus visiter d'autres noeuds du G.C.E ou lorsqu'on a visité tous les noeuds correspondant à l'état final. Si c'est le cas, une G.C.E-connexité à été trouvée. Le nombre de pas de diffusion dans cette structure permet de déduire le nombre minimal de mouvements de transit, de mouvements de transfert et d'opérations de prise et de pose nécessaires à la résolution du problème de manipulation multi-robots avec les roadmaps déjà calculées. Ce nombre est minimal car la G.C.E-connexité hérite de la faiblesse de la connexité faible : elle est une condition nécessaire à la découverte d'une solution mais pas suffisante, les roadmaps étant construites sans tenir compte des interactions géométriques négatives pouvant exister entre les robots et les objets.

Sur la figure IV.7, la propagation se passerait comme suit :

– Au départ le seul noeud accessible du G.C.E est "Cti". Il est marqué pour indiquer qu'il a été visité. Cela correspond à un mouvement de transit.

– Deux noeuds déjà marqués peut permettre de marquer un noeud lié. C'est le cas de "Cta-1". Cela correspond à une opération de prise.

– De la même manière, le noeud "p2" peut être marqué. Cela correspond à un mouvement de transfert et une opération de pose.

– On marque ensuite le noeud "Cta-2", cela correspond à un mouvement de transit et une opération de prise.

– Pour finir, nous marquons "goal-cbox" cela correspond à un mouvement de transfert et une opération de pose.

Une solution à ce problème mettra en jeu au minimum 2 mouvements de transit, 2 mouvements de transfert, 2 opération de prise et 2 opération de pose.

La structure des G.C.E et le mécanisme de propagation tels qu'ils ont été expliqués ont été simplifiés par rapport à l'implémentation réelle et ceci pour alléger le propos. Plus de détails pourront être trouvés dans [Gravot 04].

IV.3.2 Discussion

Tous les outils sont réunis ici pour résoudre le problème majeur amené par nos définitions du chapitre III : celui de trouver au moins une configuration appartenant à CS_s pour un état s. Nous pouvons déjà dans un premier temps trouver des configurations candidates pour CS_s en tirant des noeuds dans les roadmaps. Un exemple de calcul de configurations candidates a été donné dans ce chapitre (cf. IV.1.3).

Toutefois, il nous faudra vérifier qu'au moins un de ces candidats est réellement atteignable dans les contextes géométriques courants possibles. Autrement dit, il nous faudra valider les chemins dans les roadmaps en prenant en compte cette fois les inter-

actions négatives entre les chemins locaux (i.e. les arêtes des roadmaps) et les autres robots et objets. Ce travail peut-être très combinatoire. Heureusement nous avons à notre disposition une arme pour ne calculer les combinaisons que quand elles sont strictement nécessaires. Cette arme sera le calcul au préalable de plans possibles. La combinatoire que nous calculerons effectivement ne sera en fait qu'une petite partie de celle que nous aurions dû calculer si nous avions pris en compte tous les degrés de liberté du système en même temps.

Autre problème, comment guider la recherche ? Comment faire pour ne pas construire des roadmaps possédant trop de noeud tout en étant pertinentes pour la résolution d'un problème donné ? Nous proposons une stratégie pour répondre à ces questions dans le chapitre suivant.

Chapitre V

aSyMov : un planificateur

Ce chapitre présente l'algorithmique permettant de résoudre les problèmes présentés dès le début de cette thèse (cf. I.2) et formalisés en III.3.2. Cette algorithmique se présente sous la forme d'un planificateur nommé **aSyMov** (**a Sy**mbolic **Mov**e3D) [Cambon 04]. aSyMov utilise les bibliothèques de fonction d'un planificateur de tâches et celles d'un planificateur de mouvements. Ce planificateur général n'est certainement pas le meilleur possible pour les problèmes que nous proposons. C'est une solution possible, et nous parlerons tout au long de ce chapitre des qualités et des défauts des différentes parties de l'algorithme. Une évaluation du système sera faite au chapitre VII.

L'idée directrice dans la conception d'aSyMov est la suivante : trouver un plan solution en ne partant de rien. Autrement dit, au début de la planification, les roadmaps ne possèdent pas de noeud et nous n'avons calculé aucun plan symbolique (i.e. plan possible).

V.1 Principes généraux

Nous avons choisi de concevoir aSyMov comme un planificateur à recherche heuristique en avant dans l'espace d'état. La recherche en avant et l'utilisation de l'espace d'état (i.e. a contrario d'un espace des plans partiels) nous permettent de nous appuyer

sur une description complète à chaque étape de recherche. C'est essentiel pour le calcul des chemins. Nous utilisons une notion d'état particulière dans le sens où elle doit intégrer des connaissances géométriques et symboliques.

Dans cette partie, nous donnons la définition implémentée de l'état que nous utilisons. Nous donnerons ensuite une idée générale de l'algorithme du planificateur aSyMov. L'implémentation réelle sera détaillée dans une seconde partie (V.2).

V.1.1 L'état d'aSyMov

L'état utilisé par aSyMov que nous appellerons *état d'aSyMov* peut se décomposer en trois parties. Une illustration en est donnée sur la figure V.1. Ces trois parties sont :

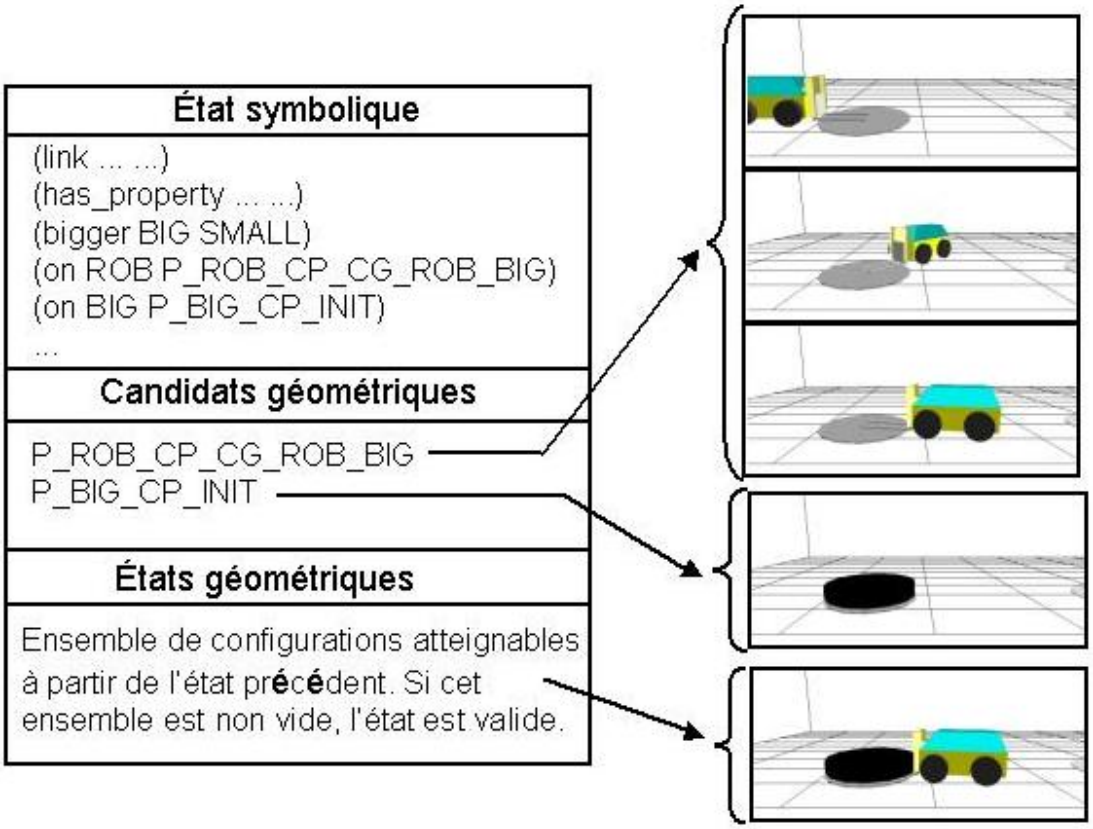

FIG. V.1 – Les trois parties d'un état d'aSyMov.

L'état symbolique : c'est cette partie qui rassemble les faits symboliques. Nous

avions appelé cette partie sym_s dans le chapitre III. Ces états symboliques correspondent exactement à ceux utilisés par un planificateur de tâches.

Les configurations candidates : nous stockons dans cette partie l'ensemble des configurations candidates pour appartenir à CS_s. Ces configurations sont candidates car elles appartiennent à CS_s^{def} et parce qu'elles appartiennent à CS_s^{reach} par connexité faible. Elles ne sont que candidates car le planificateur n'a pas encore vérifié qu'elles sont atteignables réellement sans collisions en prenant en compte les interactions géométriques négatives. Un exemple de calcul des configurations candidates a été donné plus tôt (cf. IV.1.3).

Nous rappelons qu'une configuration de *CS* est donnée par un vecteur de configurations issu des roadmaps appartenant aux robots ou aux objets actifs de l'état. Nous stockons ici des listes de noeuds pour chacune de positions des robots/objets actifs de l'état. L'ensemble des configurations candidates est obtenu en choisissant une configuration dans chacune de ces listes. Sur la figure V.1, l'état représenté met en jeu un robot actif : un chariot, et un objet actif : le gros cylindre. Chacune de leur position respective est liée à des noeuds de roadmap candidats.

Les positions à propriété de généralité n'ont pas de propriété particulière à vérifier. Seules les CS_s^{reach} ont permis de déterminer leurs candidats. Au niveau de l'implémentation, ces positions seront liées à des composantes connexes (faibles) des roadmaps. En effet, les configurations candidates, pour être atteignable à partir d'une ou de plusieurs autres configurations sans autre contraintes particulières, sont bien les composantes connexes des configurations de l'état précédent. Inversement, les positions à propriété de lien ou de zone, elles, seront lié à des ensembles de noeud. En effet il est rare que tous les noeuds d'une composante connexe satisfassent une même propriété. Nous verrons des exemples de ceci dans l'exemple développé plus loin (cf. V.2.5)

Les états géométriques : Un état géométrique est une instanciation géométrique valide de l'état symbolique. C'est une configuration de *CS* atteignable sans collisions depuis un des états géométriques de l'état précédent. Par récurence, c'est une configuration atteignable depuis l'état initial. Un état géométrique est une configuration candidate dont on a validé qu'il est bien atteignable. Un état géométrique est une configuration de CS_{free}, c'est donc un vecteur de configurations issues des différentes roadmaps appartenant aux robots ou aux objets actifs de l'état. Cette partie de l'état était appelée CS_s dans le chapitre III. CS_s peut être un ensemble infini de configurations, donc un état d'aSyMov peut avoir plus d'un état géométrique. En reprenant les définitions posées au chapitre III, nous dirons d'un état d'aSyMov qu'il est valide s'il possède au moins un état géométrique.

V.1.2 Idée générale de la recherche de plan solution

La figure V.2 illustre l'idée générale de la recherche du planificateur aSyMov.

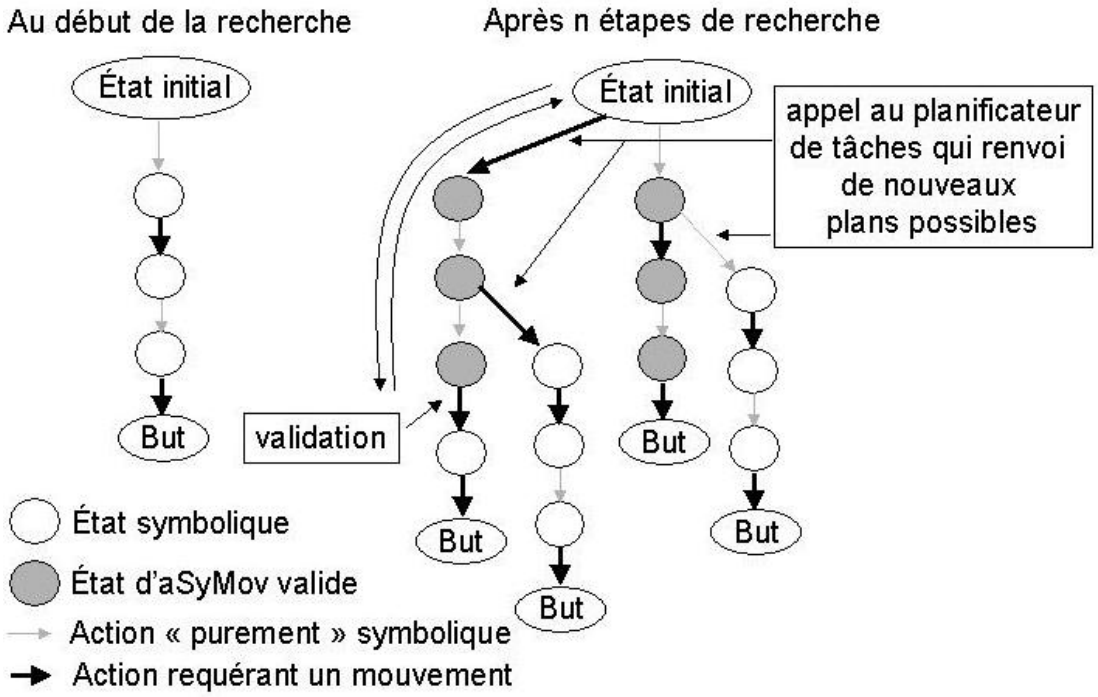

FIG. V.2 – Illustration de la recherche d'aSyMov.

Un planificateur de tâches propose des plans possibles, c'est à dire des plans respectant les contraintes relationnelles. Un planificateur de mouvements tente de valider les mouvements que requièrent ces plans. En cours de recherche, le planificateur général, aSyMov, peut toujours décider d'améliorer sa connaissance de la géométrie. Pour ce faire, il ajoutera de nouveaux noeuds aux roadmaps. Cela aura pour conséquence d'augmenter le nombre de configurations candidates des états d'aSyMov.

Au début de la recherche, les roadmaps sont vides. Durant la première étape de recherche, aSyMov appelle un planificateur de tâche qui tente de calculer un plan symbolique sur le problème symbolique associé au problème général.

aSyMov va parcourir ce plan symbolique jusqu'à ce qu'il rencontre une action symbolique requérant le calcul d'un mouvement. Les actions requérant un mouvement sont celles qui ont des instances du prédicat "`on`" dans leurs effets. A chaque état symbolique rencontré, le planificateur aSyMov :

– Crée l'état d'aSyMov correspondant à l'état symbolique. Vu qu'aucun mouvement n'a eu lieu, les nouveaux états d'aSyMov héritent des parties des "configurations candidates" et des "états géométriques" de l'état précédent

– Peut appeler le planificateur de tâches qui renvoi un ou plusieurs nouveaux plans symboliques calculés en partant de cet état symbolique et en partant d'une des actions applicable de cet état (voir sur la figure V.2). Nous verrons que dans notre implémentation réelle, cette étape est faite de manière systématique.

– Peut choisir également de développer les roadmaps pour acquérir une meilleure connaissance de la géométrie. Le planificateur cible un ou plusieurs mouvements qu'il aura peut être à calculer et ajoute des noeuds aux roadmaps correspondantes. Cette partie est détaillée en V.2.6.

A la première action requérant un mouvement rencontré sur ce plan, aSyMov créé la partie des configurations candidates pour l'état résultant. Les configurations candidates sont construites à partir des configurations candidates de l'état précédent.

Le planificateur doit maintenant valider cet état en lui trouvant une instanciation géométrique valide des positions des robots actifs de l'état. Il doit chercher parmi les configurations candidates un vecteur de noeuds correspondant à une configuration atteignable. Cette recherche s'arrête au premier état géométrique trouvé.

Vu que les configurations candidates de l'état courant sont construites à partir des configurations candidates des états précédents, il est possible de devoir valider certaines de ces configurations candidates des états précédents. Ce mécanisme ajoutera des états géométriques aux états d'aSyMov précédents. Cela peut être vu comme un retour arrière sur les précédentes instanciations géométriques. Nous détaillerons ceci en V.2.5 à travers un exemple.

Si le système à réussi à valider cette action, l'algorithme se déroule comme précédemment. Dans le cas inverse, nous choisissons un état du front de recherche, puis nous choisissons ou nous construisons un plan symbolique partant de ce nouvel état et déroulons le même algorithme (voir en V.2.4 pour les explications des choix du planificateur). Lorsqu'un état satisfaisant le but est atteint, la planification a réussi.

Cet algorithme[1] possède quelques avantages. Le premier point positif est la sauvegarde d'une certaine complétude de la recherche. Si on utilise un planificateur de tâche complet, tous les plans symboliques possibles en partant de l'état initial peuvent être explorés. De plus, nous allons voir que nos choix d'actions et d'états à explorer sont probabilistes, nous aurons donc toujours une chance de revenir sur un état du front de recherche. Le planificateur de mouvement, lui, est est complet en probabilité dans le cas de la planification de mouvement simple. aSyMov hérite a priori de cette propriété. Si une solution existe, aSyMov la trouvera au bout d'un temps qui peut être infini. Le

[1] Nous décrirons tous nos algorithmes par du texte et des figures.

deuxième point positif est le développement des noeuds de roadmap pendant la recherche. Les plans symboliques développés peuvent nous indiquer où sont les besoins en connaissances géométriques. Par exemple, le planificateur pourra se rendre compte qu'il lui faut impérativement trouver une configuration où un robot peur saisir un objet précis.

De plus le fait de développer des noeuds dans le but de résoudre un problème de planification de mouvements peut aider à résoudre d'autres problèmes de planification de mouvements d'un autre plan symbolique. C'est une des propriété des roadmaps lorqu'elles sont contruites avec une PRM.

Les points négatifs de l'algorithme reposent sur l'explosion du front de recherche. Premièrement, aSyMov peut ouvrir de manière exhaustive toutes les "voies" symboliques, il va donc apparaître de plus en plus d'état à évaluer. Généralement, le nombre des plans symboliques solutions au problème de planification de tâches associé est infini car nous autorisons l'existence de boucles dans ces plans. C'est à dire que ces plans peuvent passer plusieurs fois par le même état symbolique. En l'état de notre représentation, autoriser ces boucles est indispensable. Une action de prise puis de pose et de reprise formera une boucle dans l'espace d'état mais peut amener dans une configuration géométrique permettant de résoudre le problème. Prenons un exemple pour clarifier cet aspect. Imaginons qu'un robot doit placer un objet sur une table et que initialement cet objet est sous cette même table. Le robot devra dégager l'objet dans un premier temps, puis devra le ressaisir avec une prise différente afin de le poser sur sa configuration finale. Le plan symbolique associé passera deux fois par le même état symbolique décrivant le robot saisissant l'objet. Cette boucle est toutefois indispensable à la réussite du plan. Cela est dû principalement à notre représentation symbolique trop grossière : nous représentons par exemple deux prises différentes par la même position. Ceci pourrait être amélioré par des travaux futurs non triviaux (cf. VIII.1.3).

Deux états symboliques identiques ne produiront donc pas deux états d'aSyMov identiques. Les états d'aSyMov ne caractériseront pas le même sous-espace de *CL*. Pour s'en convaincre, prenons un autre exemple. Soit deux robots, le plan faisant se déplacer le robot 1 puis le robot 2 et le plan faisant se déplacer le robot 2 puis le robot 1 atteignent symboliquement le même état. Or les états d'aSyMov résultant sont différents. Ceci est illustré sur la figure V.3. Bien que ce problème est classique en planification de mouvements, il est gênant pour nous. Nous comprenons alors qu'un état est décrit non seulement par sa partie symbolique mais également par le plan duquel il provient.

Il y a un dernier défaut venant encore du front de recherche. On ne retire de celui-ci que les états dont on a évalué toutes les actions applicables pour former des états d'aSyMov valides. Autrement dit, tous les états ayant une action applicable nécessitant

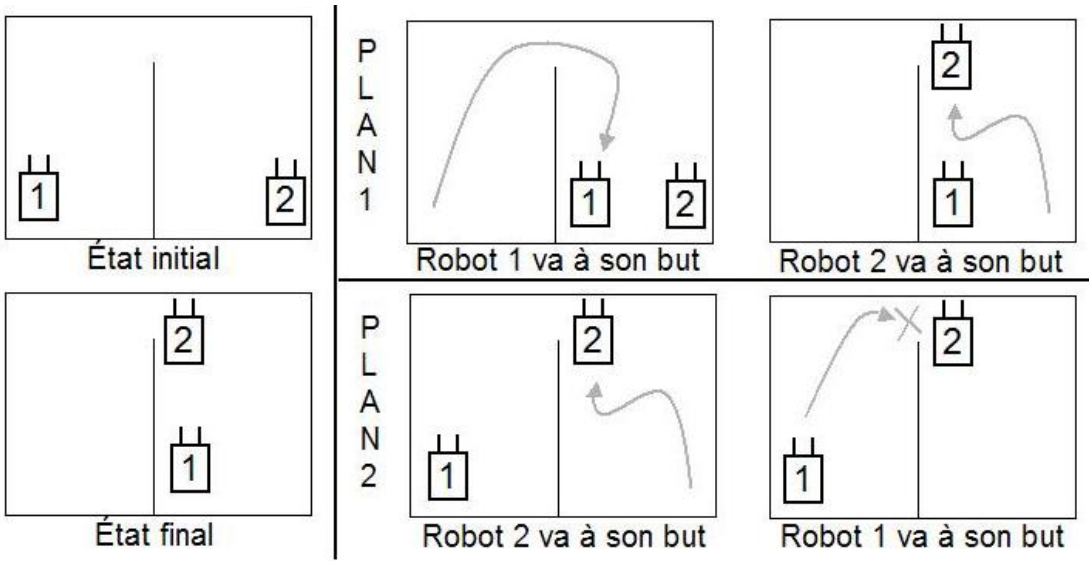

FIG. V.3 – Deux plans menant au même état symbolique ne mènent pas aux même états d'aSyMov.

un mouvement que nous n'avons pas validée resteront dans ce front de recherche. Il est possible qu'un ajout de noeuds de roadmap permette de valider ce mouvement à l'avenir. Si ce mouvement est intrinsèquement irréalisable, cet état "polluera" le front de recherche jusqu'à la réussite de la planification. Il nous faudra trouver un moyen de ne pas tenter trop souvent de valider cet état.

V.2 Implémentation

Nous décrivons ici le planificateur tel qu'il est implémenté. Nous commençons par décrire les entrées du système. Ensuite nous décrivons l'architecture logicielle d'aSyMov basée sur l'utilisation de deux bibliothèques de planificateurs. Nous décrivons ensuite les algorithmes généraux d'aSyMov. Le reste de cette partie explique en détail les composantes algorithmiques importantes.

V.2.1 Définir un problème

La figure V.4 résume les données d'entrée du système. Définir un problème consiste à définir cinq fichiers.

Le fichier "3D" décrit le monde statique tridimensionnel ainsi que les formes tridimensionnelles des différents corps qui composent un robot ou un objet. Ce fichier décrit également les articulations mécaniques entre ces différents corps (i.e. une description des degrés de liberté) et les méthodes locales des robots. Ce fichier est celui pouvant être utilisé par un planificateur de mouvements classique.

Le fichier "roadmaps" décrit :

– Les chaînes cinématiques des robots pour les calculs en cinématique directe ou inverse (cf. IV.1.2).

– Les compositions des robots et des objets. Elles sont définies comme les articulations des robots.

– Les roadmaps disponibles et leurs liens.

– Des fonctions de génération de configuration.

Ce fichier attribue de plus un identificateur pour chaque robot, pour chaque objet, pour chaque composition d'un robot et d'un objet, pour chaque roadmaps et pour chaque fonction de génération de configuration. Ce fichier est celui pouvant être utilisé par un planificateur de manipulation multi-robots utilisant la méthode des roadmaps hétérogènes (cf. chapitre IV).

Les fichiers "symboliques", un fichier de domaine qui décrit les opérateurs et un fichier problème décrivant l'état initial et l'état final. Les descriptions sont en PDDL2.1. Ces fichiers peuvent être utilisé par n'importe quel planificateur de tâches classique.

Le fichier "liens" encode les quatre relations externes $\mathscr{R}_{robot}$, $\mathscr{R}_{obj}$, $\mathscr{R}_{space}$ et $\mathscr{R}_{property}$. Les relations $\mathscr{R}_{robot}$ et $\mathscr{R}_{obj}$ sont obtenus en mettant en relation les identificateurs des robots et des objets utilisées dans le fichier "roadmaps" avec les symboles de robots et d'objets utilisés dans les fichiers symboliques. La relation $\mathscr{R}_{space}$ est obtenue en mettant en relation les symboles de sous-espaces avec les identificateurs des roadmaps spécifiés dans le fichier "roadmap" : chaque sous-espace est relié à sa roadmap le caractérisant. La relation $\mathscr{R}_{property}$ associe chaque symbole de propriété à un identificateur d'une fonction de génération de configuration du fichier "roadmaps". En effet, nous n'utilisons pas de fonctions de reconnaissance de propriété pour une configuration, mais nous générons ces configurations puis nous les "marquons" avec la propriété.

Sur la figure V.4, la partie grisée correspond aux redondances : les fichiers symboliques représentent de manière relaxée une partie du fichier "roadmap".

Ces données en grisé de la figure V.4 sont délicates à modifier. Il faut faire des allers retours entre les divers fichiers pour, par exemple, changer les liens des roadmaps. En effet, nous n'avons pas défini d'outil facile d'utilisation pour générer des problèmes aisément. Par contre, il est très facile de changer les autres informations. Par exemple nous avons généré très rapidement les variantes (1), (3) et (4) du problème de tours de Hanoï à partir de la variante (2) (cf. I.2.3). Pour la variante 1, nous avons juste supprimé

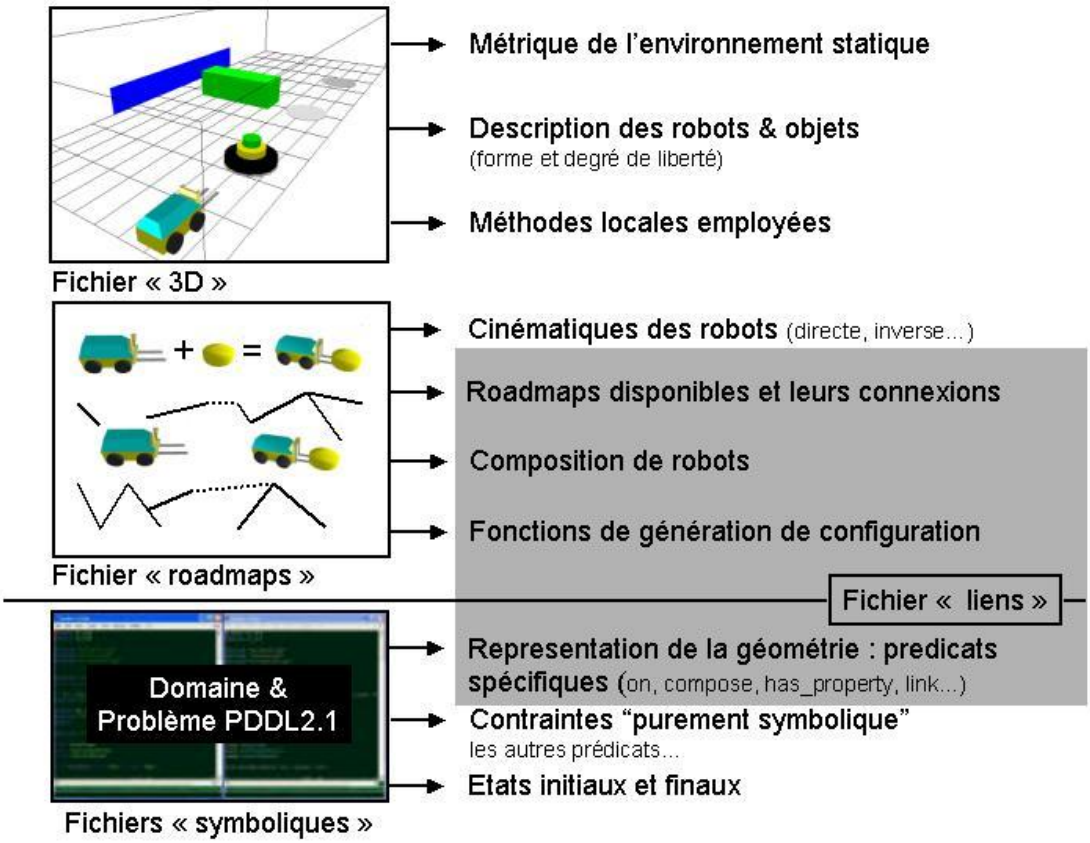

FIG. V.4 – Les fichiers utilisés pour définir un problème.

dans l'état initial du problème symbolique l'instance du prédicat "`on`" concernant le second robot. Pour les instances (3) et (4), nous avons en plus ajouter des obstacles statiques dans le fichier "3D". Nous pourrions également très facilement :

– enlever la contrainte de placement des cylindres sur un plus petit. Cette modification se ferait au niveau des opérateurs symboliques définis.

– changer le but du problème comme par exemple de mettre en but une composition de tour de deux disques sur la pile centrale. Cette modification se ferait sur l'état final symbolique dans le fichier décrivant le problème symbolique.

– changer la forme des robots en leurs ajoutant par exemple un volumineux porte bagage. Cette modification se ferait dans le fichier "3D".

– changer la méthode locale des robots, en les affublant d'une peu crédible méthode linéaire ou en les autorisant à voler à travers l'environnement. Cette modification se ferait dans le fichier "roadmaps".

Le développement d'un nouveau problème peut être relativement long, surtout pour la définition du fichier "roadmaps" (et du fichier "3D" si nous devons modéliser de nouveaux robots). Par contre une fois le modèle acquis, nous pouvons en extraire une infinité de variantes.

V.2.2 Architecture logicielle

D'un point de vue logiciel, aSyMov est basé sur le planificateur de tâches, Metric-FF [Hoffmann 03] et sur le planificateur de mouvement Move3D [Siméon 01]. Nous avons adapté ce dernier pour gérer la manipulation et la méthode des roadmaps hétérogènes.

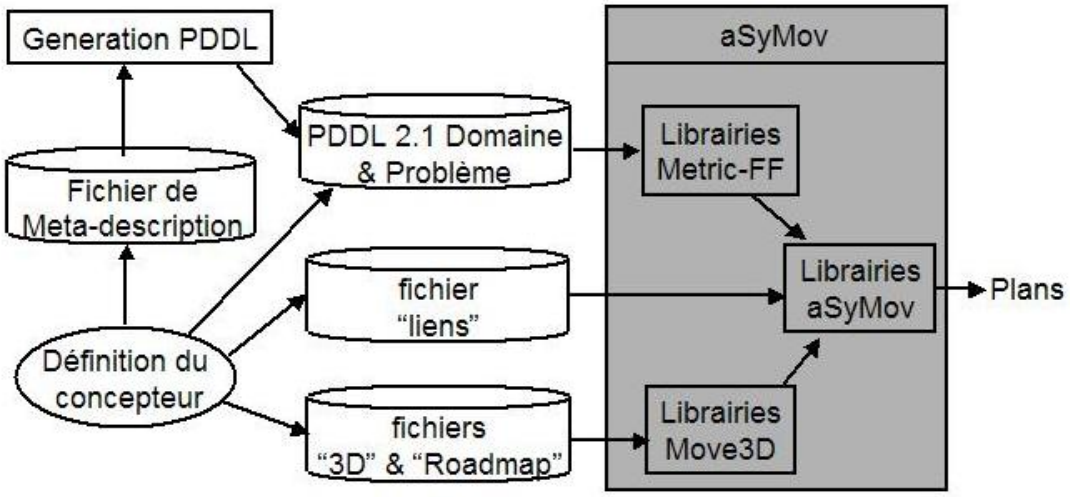

FIG. V.5 – Architecture logicielle d'aSyMov.

Les fichiers déjà décrits en V.2.1 sont représentés sur la figure V.5. Le fichier de "Meta-description" est le fichier d'entrée permettant la génération automatique des symboles (cf. III.2.3).

Les fichiers symboliques sont utilisés par les librairies de Metric-FF, le fichier "3D" et "roadmaps" par les librairies de Move3D. aSyMov lui, a accès au fichier "liens" et aux structures de données créées par Move3D et Metric-FF. Il peut ainsi construire des structures de données hybrides.

V.2.3 Algorithmes généraux

Nous avons basé aSyMov sur un algorithme simple (voir figure V.6). Après une phase de chargement, d'initialisation et de prétraitement, un état est choisi puis étendu.

Étendre un état revient à créer un nouvel état par l'application d'une action. Une fois le but atteint, le plan est extrait puis amélioré. L'extraction des plans et les améliorations qui lui sont apportés feront l'objet du chapitre VI.

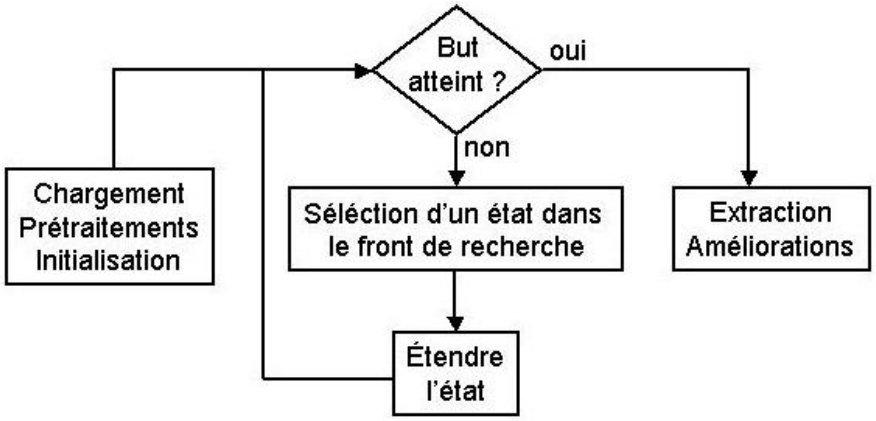

FIG. V.6 – Algorithme général d'aSyMov.

Le chargement consiste à créer les structures de données de Metric-FF et de Move3D sur la base des fichiers descriptifs.

Les prétraitements sont des calculs misent en place par des fonctions du planificateurs de tâches et par celles du planificateur de mouvements. Le planificateur Metric-FF analyse le domaine et le problème pour retirer les faits inutiles et instancier les actions nécessaires. Nous avons séparé la partie des prétraitements de la résolution des problèmes. Les prétraitements symboliques ne seront faits qu'une fois alors que aSyMov fera plusieurs requêtes de résolution de problèmes de planification de tâches. Le planificateur Move3d crée des structures pour simplifier notamment le travail du détecteur de collisions.

L'initialisation créera des structures hybrides à la fois reliées à des termes symboliques et reliées à des entités géométriques (les positions, les robots, les sous-espaces...). Le planificateur cherchera à tirer quelques noeuds dans les roadmaps pour chacune des positions du problème. S'il échoue, nous considérerons que le problème n'admet pas de solution. Cela nous permet de détecter les cas de collisions dans les états initiaux ou finaux ou les cas où les objets sont inatteignables par les robots.

L'initialisation indiquera également si des contradictions apparaissent dans les données d'entrée. Par exemple si les fichiers "symboliques" indiquent une composition entre un robot et un objet qui n'apparaît pas dans le fichier "roadmaps".

Le coeur d'aSyMov est la procédure d'extension d'état illustré sur la figure V.7.

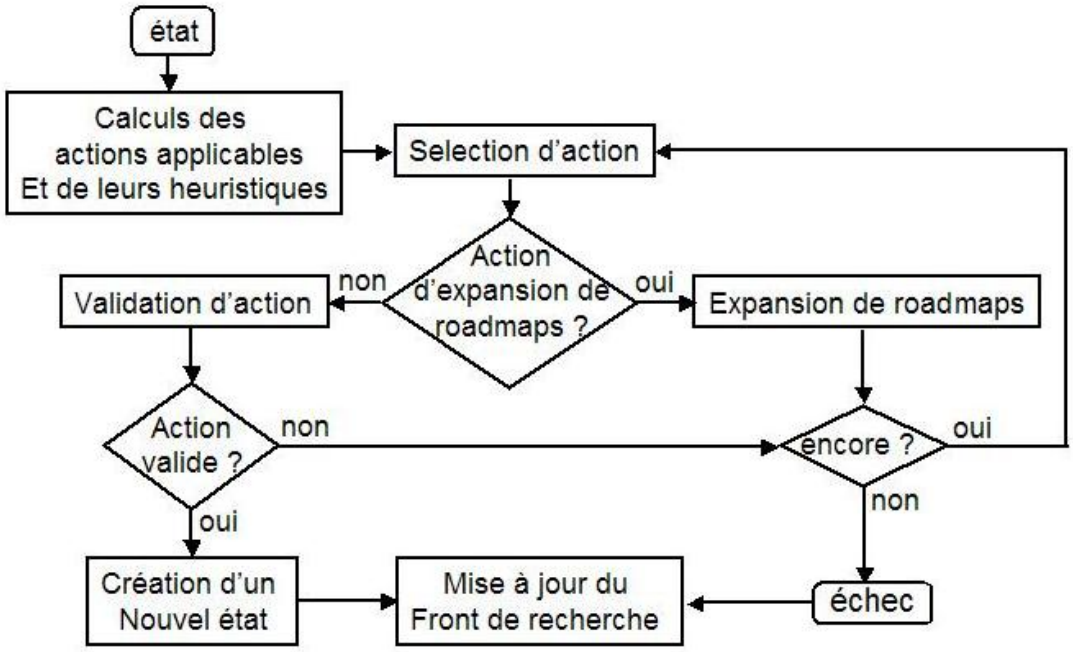

FIG. V.7 – l'algorithme d'extension d'état.

A partir de l'état à étendre, nous calculons les actions applicables sur la base de la partie "état symbolique" de l'état d'aSyMov. Nous ajoutons à ces actions des actions dites *actions d'expansion de roadmaps*.

Nous calculons des coûts pour chacune des actions puis nous sélectionnons l'une d'elle (cf. V.2.4 pour les choix du planificateur). Si cette action est une action applicable, nous tentons de la valider en trouvant au moins un état géométrique valide pour l'état résultant (cf. V.2.5). Si cette action est une action d'expansion de roadmaps, nous ajoutons des noeuds aux roadmaps (cf. V.2.6.

La procédure de sélection d'une action et de son application peut être répétée un nombre de fois paramétrable tant qu'un nouvel état n'a pas été ajouté. Si ce paramètre est de 4, on peut avoir par exemple, deux tentatives de validation et deux actions d'expansion de roadmaps avant d'abandonner l'extension de l'état.

V.2.4 Les actions, les états et comment les choisir

Nous avons choisi de représenter les ajouts de noeuds dans les roadmaps par des actions qui seront sur un même pied d'égalité que les actions symboliquement applicables. De cette manière, la sélection de l'action représente le choix que doit faire le

planificateur entre continuer la recherche d'un plan avec le niveau de connaissance topologique qu'il a déjà acquis ou d'acquérir une connaissance plus profonde de la topologie des sous-espaces de configurations qu'il manipule.

Inventaire des actions

Le premier type d'action regroupe les actions applicables. Ce sont les actions symboliques applicables sur la partie "état symbolique" de l'état d'aSyMov. C'est à ce moment précis que nous assurons le respect des contraintes symboliques du problème. Le nombre moyen d'actions applicables est donc égal au facteur de branchement du problème symbolique. Nous remarquons que le recensement de toutes les actions symboliquement applicable revient à dévelloper autant de plans possibles que d'actions symboliquement applicable.

Le second type d'action regroupe les actions d'expansion de roadmaps[2]. Ces actions permettent d'ajouter des noeuds aux roadmaps (cf. V.2.6 pour plus de détails). Nous en avons défini deux genres :

– *Les actions d'expansion par recherche en profondeur*. Ces actions vont favoriser l'ajout de noeuds dans les roadmaps permettant de calculer les mouvements que nécessite le plan possible courant. Plus l'action représentant ce mouvement est proche de l'état d'aSyMov que l'on souhaite étendre, plus on ajoutera de noeuds dans les roadmaps concernés par ce mouvement.

– *Les actions d'expansion par recherche en largeur*. Ces actions vont favoriser l'ajout de noeuds dans les roadmaps concernées par la totalité des actions applicables à l'état d'aSyMov courant.

Pour l'état courant, il y aura autant d'actions d'expansion par recherche en profondeur que d'actions applicables. On y ajoute une action d'expansion par recherche en largeur. Au total, nous aurons donc un facteur de branchement moyen égal à $2 * (\textit{Facteur De Branchement Symbolique Moyen}) + 1$ actions. Il nous faut donc une méthode pour choisir au mieux une de ces actions.

Heuristique principale

Nous rappelons que notre représentation symbolique est une relaxation du vrai problème. La relaxation est de ne pas prendre en compte les préconditions et conséquences géométriques des actions. La longueur du plan symbolique en partant d'une action est donc une heuristique tout à fait intéressante. C'est même une heuristique admissible, mais vue la spécificité du front de recherche, cette propriété ne sera pas utilisée.

[2]Nous appelions précédemment ces actions "actions d'apprentissage" dans le sens où elles permettent d'en apprendre plus sur la topologie. Ce terme semble toutefois délicat à utiliser dans ce cas...

Nous devons donc calculer un plan symbolique pour chaque action applicable. Les problèmes que nous résolvons sont, en général, plus complexe en terme de géométrie qu'en terme de planification de tâche. Dans les domaines que nous avons développé le temps de calcul passé dans la résolution des problèmes symboliques est négligeable par rapport au temps de calcul passé dans le détecteur de collisions[3].

Toutefois nous avons optimisé le nombre d'appels au planificateur symbolique en stockant dans une structure à accès rapide (une table de hachage) les états symboliques déjà rencontrés lors des calculs de plan symbolique. Lors d'une requête de calcul de plan, si l'état initial de la requête a déjà été rencontré, nous extrairons directement le plan symbolique de notre structure mémorisé. Ceci permet un gain significatif dans le nombre d'appels au planificateur symbolique.

Cette optimisation permet également de se représenter la "topologie" des plans symboliques. Nous nous en servirons pour notre outil de visualisation de l'avancement de la recherche (voir figure VII.3 et VII.4).

Nous avons également en notre possession un outil permettant d'analyser les connexités entre les roadmaps. Cet outil est basé sur une analyse du Graphe des Connexités Élémentaires que l'on a déjà évoqué en IV.3.

Notre heuristique principale est une intégration de ces deux techniques. L'heuristique principale pour une action applicable sera obtenue comme suit.

Première étape Le planificateur calcul la partie des "configurations candidates" de l'état résultant. Si le planificateur ne trouve aucun candidats, cette action sera rejetée du choix.

Deuxième étape Si l'action n'a pas été rejetée du choix dans la première étape, aSyMov envoie une requête au planificateur de tâches qui produit un plan symbolique en prenant comme première action l'action applicable que l'on veut évaluer. S'il n'existe pas de tel plan symbolique, l'action est rejetée du choix également. En cas de réussite, la longueur du plan symbolique nous donne une première estimation heuristique que nous allons corriger dans la troisième étape.

Troisième étape Si l'action n'a pas été rejetée par la deuxième étape, le planificateur vérifie que le plan dans son ensemble est valide au niveau du G.C.E. en diffusant les "configurations candidates" dans le G.C.E. Il est possible que pour une action de ce plan, cette diffusion ne puisse pas se faire en une seule étape. Cela peut être le cas par exemple pour une action concernant une composition de robots qui doit mener cette composition de robots d'une position à une autre. Il est alors possible que la position finale soit atteignable mais seulement avec des actions de repose puis de re-

[3]Évidemment, nous pourrions toujours proposer à aSyMov un problème très ardu en terme d'ordonnancement avec une géométrie inexistante, dans ce cas notre planificateur ne serait vraiment pas performant. En effet nous calculerions beaucoup trop de plan symbolique alors que le premier calculé aurait été suffisant.

prise. Le plan symbolique n'est alors pas en conformité avec la topologie actuellement connue. Nous corrigeons alors cette longueur de plan du nombre de pas de diffusion supplémentaires nécessaires. Ce nombre de pas supplémentaires est toutefois corrigé suivant la profondeur de l'action concernée dans le plan symbolique. Plus cette action est profonde plus nous réduirons ce nombre de pas supplémentaires. Nous prenons en compte le fait qu'une expansion de roadmap puisse ouvrir une "voie" plus simple pour cette action.

En résumé, aSyMov calcule son heuristique principale en calculant la longueur d'un plan symbolique qu'il corrige grâce aux connaissances topologiques déjà acquises. En d'autres termes, l'heuristique principale mesure la longueur d'un plan en relaxant les collisions qui peuvent survenir mais en ne relaxant pas les impossibilités structurelles de la topologie connue couramment. Par exemple notre heuristique détectera et guidera avantageusement dans les situations suivantes :

– Si l'espace de travail est partagé en deux par un mur, cela sera détecté par une analyse du G.C.E et donc pris en compte par l'heuristique générale.

– Si un robot doit faire une prise puis repose puis reprise d'un objet.

– Si on doit atteindre une série de positions pour appliquer des actions symboliques.

Il existe un cas de figure qui piège l'heuristique : si un objet est gênant et s'il n'a pas a priori à être déplacé dans le plan. Dans ce cas de figure, le guidage va être moins efficace et il faudra donc tempérer les conseils de l'heuristique générale par d'autres coûts. C'est ce que nous expliquons ci-après.

Coût des actions, coût des états

L'heuristique générale est insuffisante pour éviter les pièges évoqués ci-dessus. Le planificateur pourrait se bloquer en voulant absolument valider une action ou ajouter trop de noeuds dans les roadmaps pour un mouvement intrinsèquement infaisable. C'est le cas si le planificateur tente via une action de faire rejoindre une configuration à un robot et si tous les chemins possibles du robot pour la rejoindre sont obstrués par un objet.

Nous avons décidé de pondérer l'heuristique générale par une prise en compte d'autres coûts basés notamment sur le nombre d'échecs que le planificateur subi sur la validation d'une action. Ces coûts ont été paramétrés empiriquement sur le jeu de problèmes que nous avions à notre disposition. Ces coûts sont donc hautement discutables.

Le coût total d'une action applicable et les actions d'expansion de roadmaps auront trois coûts que nous sommons :

Coût total = Coût propre + Coût heuristique + Coût des échecs
avec :

– **Coût propre** : C'est le coût donné arbitrairement par le concepteur du problème aux actions. Il peut permettre de favoriser ou de défavoriser certaines actions. Dans notre implémentation actuelle nous donnons aux actions applicables un coût propre de 1 et aux actions d'expansion de roadmaps un coût de 2. Nous voulons favoriser la recherche d'une solution plutôt que l'augmentation des connaissances sur la topologie. En effet cela permet à aSyMov de privilégier d'abord l'avancement dans la recherche par la validation des actions.

– **Coût heuristique** : c'est l'estimation de l'intérêt de l'action. Ce coût est égal à l'heuristique principale. Pour les actions d'expansion de roadmaps à recherche en profondeur, ce coût est égal à celui de l'action applicable associée. Pour l'action d'expansion de roadmap à recherche en largeur, le coût est la moyenne des coûts des actions applicables.

– **Coût des échecs** : ce coût doit nous permettre de pas tenter de valider trop souvent la même action applicable dont le mouvement associé est intrinsèquement infaisable. On le calcule avec une fonction linéaire du nombre d'échecs dans les tentatives de validation de l'action. Pour les actions d'expansion de roadmap, ce coût doit permettre de limiter les ajouts sur une roadmap que l'heuristique générale aurait tendance à vouloir favoriser. Ce coût est calculé avec une fonction linéaire du nombre de fois où cette action a été sélectionnée.

Nous devons également être capable de choisir les états pour déterminer le "meilleur" état à étendre. Nous leurs attribuons un coût qui est également la somme de trois coûts :

– **Coût propre** : coût du plan nécessaire pour atteindre cet état depuis l'état initial. C'est la longueur du plan en nombre d'actions déjà parcourues.

– **Coût heuristique** : c'est la somme des coûts des actions applicables partant de cet état non encore validées.

– **Coût des échecs** : obtenu avec une fonction linéaire du nombre d'échecs que l'on a eu en tentant d'étendre cet état. Nous avons également ajouté un coût proportionnel au nombre d'état d'aSyMov possédant la même partie "état symbolique". En effet, nous avons remarqué empiriquement que des même états symboliques mènent relativement souvent à des états d'aSyMov identiques, ce coût permet de ne pas sélectionner trop souvent des états semblables qui ont de bonnes chances d'être sélectionnés mais qui ne feront pas partie du plan solution.

Sélection d'action

Nous possédons maintenant de quoi comparer ces actions. Nous introduisons un processus de choix non déterministe. La sélection d'une action ou d'un état se fera en

probabilité : plus une action ou un état est “bien noté” plus il aura de chance d’être sélectionné. Toutefois, lorsque l’action applicable au coût le plus faible est une action applicable “purement symbolique”, elle sera sélectionnée sans faire de choix probabiliste. Ceci revient à dérouler le plan symbolique jusqu’à rencontrer une action applicable à conséquences géométriques.

Nous ne comprenons pas totalement pourquoi nous obtenons de meilleurs résultats moyens avec cette méthode qu’avec des sélections déterministes. Nous pensons que la plupart de nos domaines qui nous ont permis de régler nos coûts font interagir fortement la géométrie avec les contraintes symboliques. Notre heuristique principale nous donne alors une mesure relativement éloignée de la mesure idéale, les choix probabilistes représentent notre ignorance sur la qualité de cette heuristique.

Nous proposons une évaluation de cette gestion de coûts et de sélection d’action en VII.1.1.

V.2.5 Validation : Instanciation géométrique d’une action

Principe général

Lorsque l’action sélectionnée est une action symboliquement applicable (et donc pas une action d’expansion de roadmap), il faut valider cette action. En d’autres termes, il faut trouver une configuration atteignable pour chacune des positions des robots actifs de l’état résultant. C’est ce que nous appelons aussi l’instanciation géométrique d’une action ou d’un état. C’est à ce moment précis que l’on garantie le respect des contraintes géométriques.

Si cette action symboliquement applicable est une action “purement symbolique”, la validation de cette action est simple. Le planificateur crée simplement l’état d’aSyMov résultant en lui donnant comme partie “état symbolique” l’état symbolique résultant de l’application de l’action. Les parties des “configurations candidates” et des “états géométriques” sont directement héritées de l’état précédent. En effet, il n’y a pas eu de mouvements entre les deux états.

Lorsque cette action est une action nécessitant un mouvement ou une composition, le processus de validation est appelé. Nous rappelons que pour limiter la complexité, nous construisons nos roadmaps indépendamment des configurations des autres robots ou objets. Ce processus de validation va donc nous permettre de vérifier la validité des arêtes et de noeuds de roadmaps dans le contexte courant de la géométrie. Une arête ou un noeud de roadmap peut-être valide dans un contexte donné et invalide dans un autre.

Deux principes importants sont au coeur de cet algorithme. Le premier c’est son aspect paresseux : le processus de validation s’arrête au premier état géométrique trouvé.

Le second principe, c'est que le processus pourra toujours revenir sur les instanciations précédentes pour ajouter de nouveaux états géométriques aux états d'aSyMov déjà valides (i.e. qui possèdent déjà au moins un état géométrique).

Un exemple

Nous avons choisi de présenter l'algorithme de validation à travers un exemple. Nous l'avions déjà fait dans [Gravot 03b] sur un autre exemple que celui présenté ici.

Ici, nous choisissons un problème du type "chariots élévateurs" avec un seul robot et une seule caisse (figure V.8). La description de la validation nous permettra également de montrer comment nous créons et gérons les états d'aSyMov.

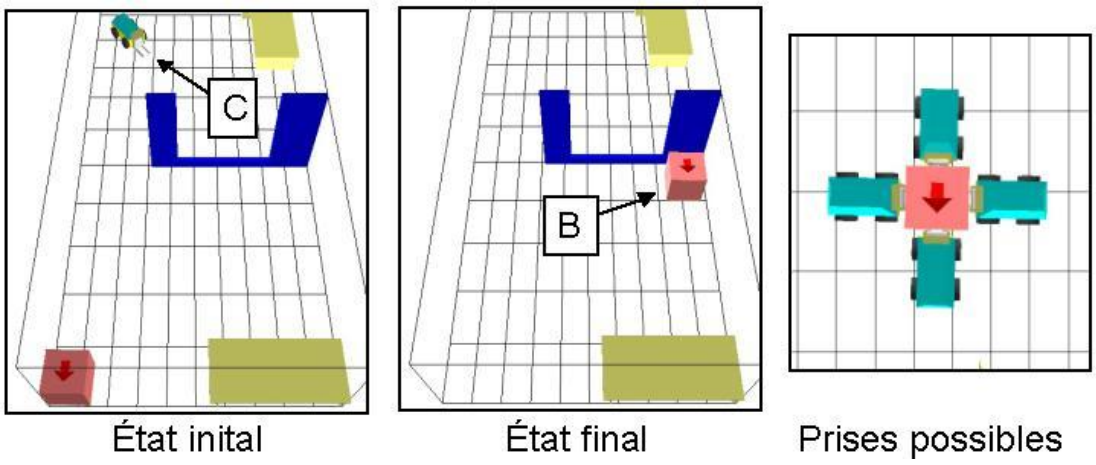

FIG. V.8 – Un problème du type "chariots élévateur".

Dans ce problème, un chariot `C` doit déplacer une boîte `B` d'un endroit à un autre de l'environnement. Nous rappelons que la boîte `B` est orientée, ceci est représenté par une flèche sur la figure V.8. Nous avons défini quatre prises possibles pour saisir l'objet. Trois roadmaps sont définies : une roadmap de transit pour les mouvements du robot seul, une roadmap de transfert pour les mouvements de la composition chariot-caisse `C-B` et une roadmap de placement pour la définition des placements stables de `C`. La roadmap de transit peut s'appuyer sur une roadmap relative (cf. IV.2.2) pour les mouvements d'approche de la boîte. La roadmap de transfert, elle, peut s'appuyer sur des techniques évoquées en IV.1.2 pour construire des configurations de la composition chariot-caisse `C-B` en partant d'une configuration de la caisse. La roadmap de transfert possédera au minimum quatre composantes connexes : une par prise.

La difficulté de ce problème réside dans le fait que le chariot n'est pas capable de transporter la caisse directement de sa position initiale à sa position finale. La cause en est l'orientation des configurations initiales et finales de la caisse et les bornes de l'environnement. Un plan solution à ce problème devra faire déposer la caisse en un placement intermédiaire permettant un changement de prise.

Nous n'abordons pas ici les choix des actions qui nous amènent au plan solution, ceci ayant déjà été évoqué plus haut. Nous donnons le plan symbolique que le planificateur va tenter de valider. Il est décrit ci-après avec la nomenclature donnée en III.2.3.

```
A1. GOTO C P_C_CP_INIT P_C_CP
A2. GOTO C P_C_CP P_C_CP_CG_C-B
A3. GRASP C   P_C_CP_CG_C-B
          B   P_B_CP_INIT
          C-B P_C-B_CG_CP_C
A4. GOTO C-B P_C-B_CG_CP_C P_C-B_CG
A5. GOTO C-B P_C-B_CG P_C-B_CG_CP_C
A6. UNGRASP C-B P_C-B_CG_CP_C
            C   P_C_CP_CG_C-B
            B   P_B_CP_CG_C-B
A7. GOTO C P_C_CP_CG_C-B P_C_CP
A8. GOTO C P_C_CP P_C_CP_CG_C-B
A9. GRASP C   P_C_CP_CG_C-B
          B   P_B_CP_CG_C-B
          C-B P_C-B_CG_CP_C
A10. GOTO C-B P_C-B_CG_CP_C P_C-B_CG
A11. GOTO C-B P_C-B_CG P_C-B_CG_CP_C
A12. UNGRASP C-B P_C-B_CG_CP_C
             C   P_C_CP_CG_C-B
             B   P_B_CP_GOAL
```

Remarques. Ce plan met en jeu deux opérations de prise et de pose.

Le lecteur remarquera que nous utilisons deux actions `GOTO` pour ne représenter a priori qu'un seul mouvement. Dans notre implémentation, nous forçons le planificateur pour qu'il ne puisse pas proposer d'action de `GOTO` entre deux positions à propriété de lien ou de zone. Il doit d'abord visiter une position à propriété de généralité. Les positions à propriété de généralité sont utilisés pour trouver des configurations intermédiaires au cas où un robot gêne le mouvement d'un autre robot. De plus ce passage par des positions à propriété de généralité facilite la propagation des configurations candidates. L'exemple devrait bien illustré ceci.

Nous supposons de plus que nous avons une connaissance topologique suffisante et donc nous n'aborderons pas encore l'expansion des roadmaps. La planification débute avec des roadmaps non vides qui seront suffisantes pour extraire une solution.

Représentation graphique. Pour faciliter la compréhension, nous donnons une représentation des différents états d'aSyMov explorés sur les figures allant de V.9 à V.14. Sur ces figures, la partie "état symbolique" est indiquée en dessous de chaque vignette. Nous utilisons la nomenclature proposée en III.2.3. Les états géométriques sont représentés par les dessins de chariots et de boîtes. Les configurations candidates sont représentées par des noeuds de roadmap sur les vignettes. Nous rappelons que les configurations candidates des positions à propriété de généralité sont liées à des composantes connexes alors que les configurations candidates des positions à propriété de lien ou de zone sont liées à des ensembles de noeuds.

Au départ, nous sommes sur l'état initial. Chaque position des robots actifs de cet état est reliée à un noeud unique : `(ti0)` pour `P_C_CP_INIT` et `(p0)` pour `P_B_CP_INIT` (voir figure V.9). Maintenant, pas à pas, nous allons expliquer les validations de chaque action du plan symbolique.

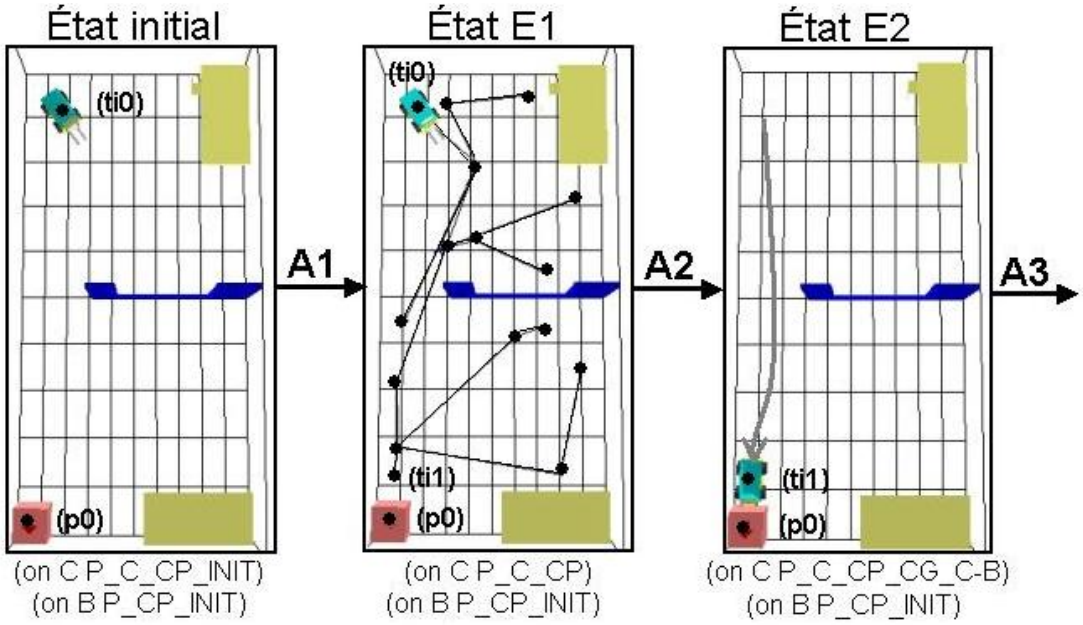

FIG. V.9 – Validation : les actions A1 et A2.

```
A1. GOTO C P_C_CP_INIT P_C_CP
```

Signification. Cette action demande au chariot C de passer de sa position initiale à sa position à propriété de généralité.

Validation. Le processus de validation est un processus paresseux. Vu que la configuration initiale du chariot possède également la propriété de généralité, il est inutile ici de calculer un mouvement pour aller chercher un autre noeud. Le nouvel état induit par cette action est différent de l'état initial par sa partie "état symbolique" et par sa partie des "configurations candidates". La position P_C_CP a comme candidat géométrique toute la composante connexe de la configuration initiale du chariot. Le nouvel état E1 possède par contre la même partie des "états géométriques" que son prédécesseur. Ceci est illustré sur la figure V.9.

```
A2. GOTO C P_C_CP P_C_CP_CG_C-B
```

Signification. Le chariot doit aller de sa position à propriété de généralité à sa position à propriété de lien possédant un lien avec la configuration initiale de la boîte B.

Validation. Du point de vue de la roadmap de transit, cette action est faisable vu que la composante connexe dans laquelle se trouve le chariot C comprend également un noeud lié à la roadmap de transfert de la composition chariot-caisse C-B lui-même lié à la configuration initiale de la boîte B. C'est le noeud (ti1). La validation est alors appelée. Elle doit vérifier que le chemin entre (ti0)et (ti1) est sans collision avec les autres robots ou objets. Le chemin est sans collision avec la boîte B. Un nouvel état, E2 est créé (voir figure V.9).

La troisième action proposée est :

```
A3. GRASP C   P_C_CP_CG_C-B
          B   P_B_CP_INIT
          C-B P_C-B_CG_CP_C
```

Signification. Le chariot doit se composer avec la boîte qui est dans sa position initiale.

Validation. Cette action ne nécessite pas de mouvement. Elle permet de passer de la roadmap de transit à la roadmap de transfert. Aucun test de collision n'est donc appelé. L'action A3 est valide et produit un nouvel état E3. Ceci est illustré sur la figure V.10.

```
A4. GOTO C-B P_C-B_CG_CP_C P_C-B_CG
```

Signification. La composition chariot-boîte doit aller à une position à propriété de généralité.

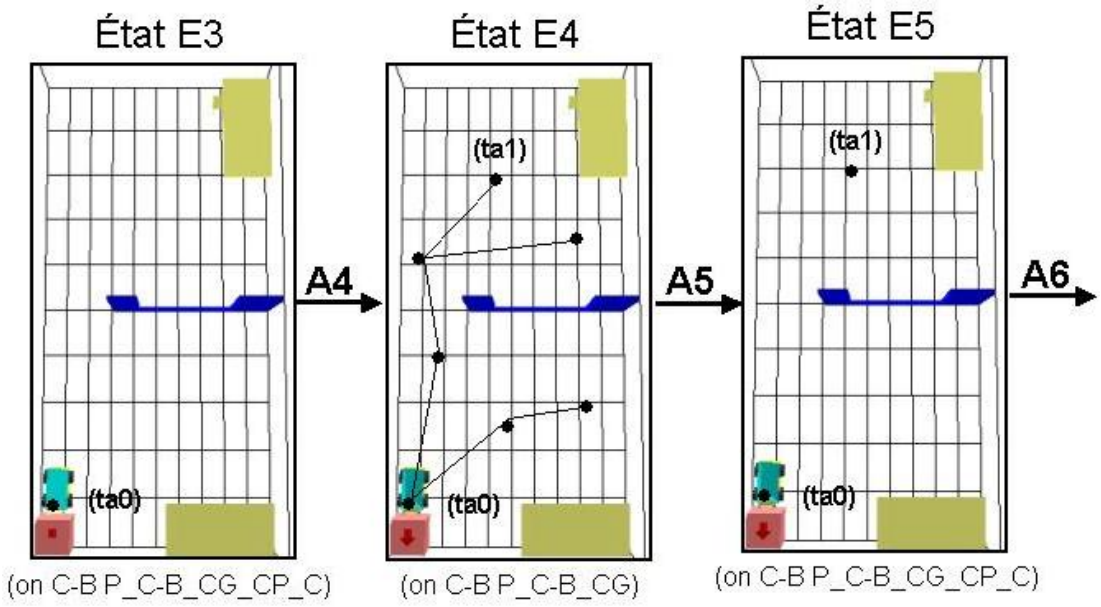

FIG. V.10 – Validation : les actions A3, A4 et A5.

Validation. La validation opère comme pour l'action `A1`. On ne cherche pas de nouvel état géométrique. On crée un état `E4` possédant comme configurations candidates toute la composante connexe du noeud `(ta0)` (voir figure V.10). Cette composante connexe correspond à la composante connexe de la roadmap de transfert où la prise discrète est constante : la flèche de la boîte est toujours vers le bas par rapport au chariot.

```
A5. GOTO C-B P_C-B_CG P_C-B_CG_CP_C
```

Signification. La composition chariot-boîte doit se rendre sur une position à propriété de lien.

Validation. Deux noeuds sont candidats : `(ta0)` et `(ta1)`. Nous sommes déjà sur le noeud `(ta0)`, aucun calcul de validation de mouvement n'est donc nécessaire. On crée un nouvel état `E5` (voir figure V.11). Dans cet état, le noeud `(ta0)` fait partie des états géométriques alors que le noeud `(ta1)` est conservé en tant que candidat géométrique.

```
A6. UNGRASP C-B P_C-B_CG_CP_C
            C   P_C_CP_CG_C-B
            B   P_B_CP_CG_C-B
```

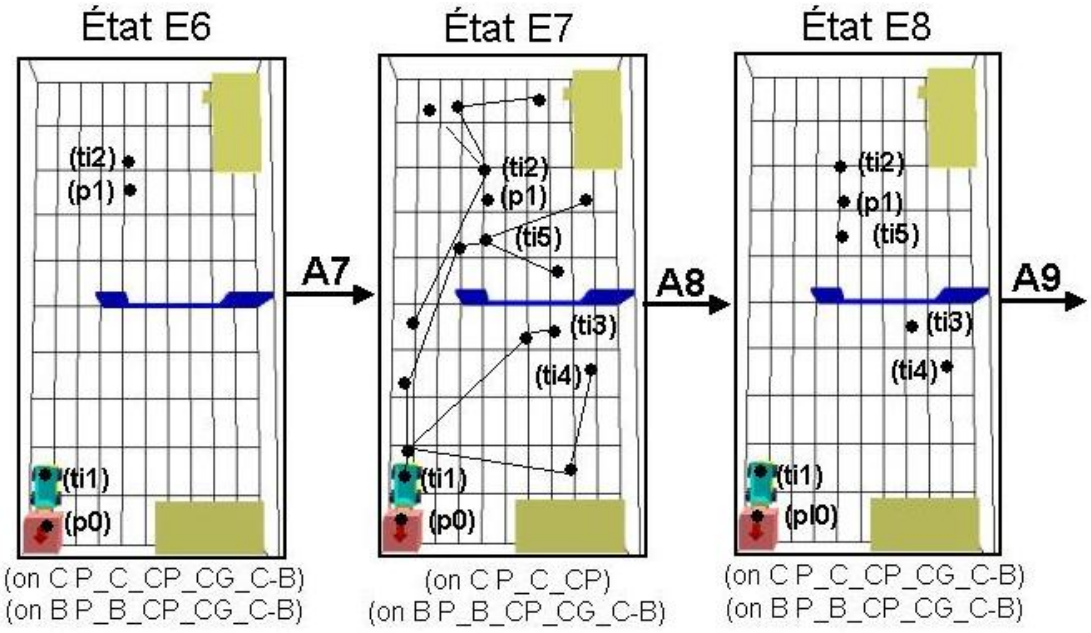

FIG. V.11 – Validation : pour les actions A6, A7 et A8, aucun mouvement n'est calculé.

Signification. C'est l'action de décomposition de chariot-boîte.

Validation. Vu qu'il n'y a pas d'autres contraintes, la décomposition peut se faire sur le noeud `(ta0)`, donc sur place. Une nouvelle fois, pas de calcul de validation de mouvement n'est nécessaire. Par contre, on peut remarquer sur la figure V.11 que les configurations candidates sont propagées.

`A7. GOTO C P_C_CP_CG_C-B P_C_CP`

Signification. Le chariot doit se rendre dans une position à propriété de généralité.

Validation. Le noeud `(ti1)` convient très bien, aucun calcul de validation de mouvement n'est nécessaire. Dans le nouvel état `E7` les configurations candidates pour le chariot sont données par la composante connexe de `(ti1)` **et** de `(ti2)`. Il se trouve que c'est la même. Les candidats géométriques pour la boîte sont `(p0)` et `(p1)`. L'action `A7` produit un nouvel état `E7` illustré sur la figure V.11.

`A8. GOTO C P_C_CP P_C_CP_CG_C-B`

Signification. Le chariot doit se rendre dans une position à propriété de lien.

Validation. Le noeud `(ti1)` convient très bien. Aucun mouvement n'est alors calculé. Les noeuds `(ti2)`, `(ti3)`, `(ti4)` et `(ti5)` étaient également candidats. Ils sont

stockés dans les configurations candidates de manière à former un nouvel état E8 (voir figure V.11).

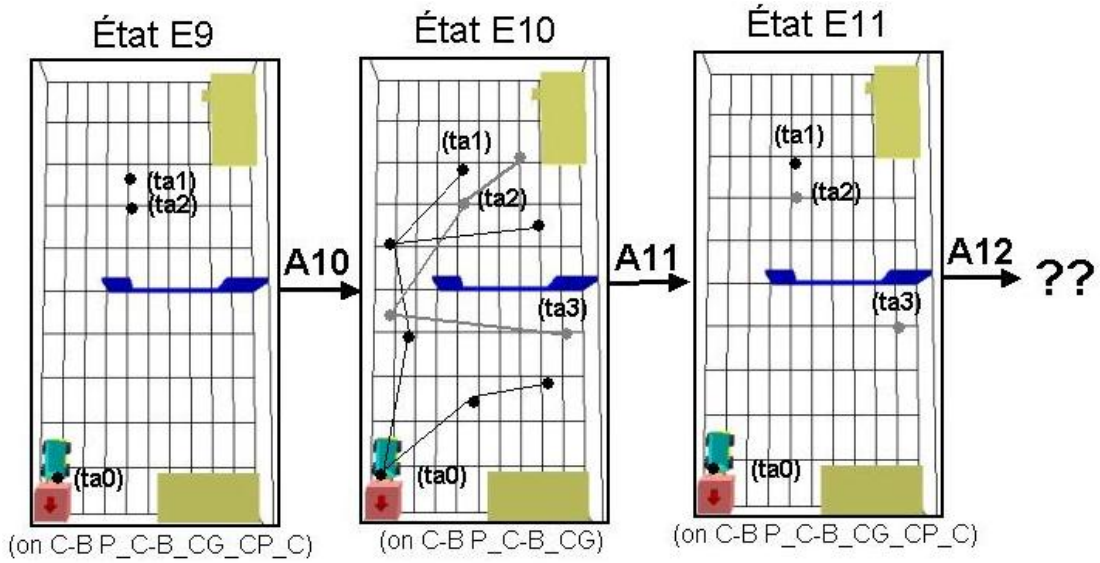

FIG. V.12 – Validation : pour les actions A9, A10 et A11, aucun mouvement n'est calculé.

```
A9. GRASP C   P_C_CP_CG_C-B
          B   P_B_CP_CG_C-B
          C-B P_C-B_CG_CP_C
```

Signification. Le chariot doit se composer avec la boîte.

Validation. L'état géométrique précédent, le couple de noeud `(ti1),(p0)` est tout à fait apte à cette opération. Aucun calcul de mouvements n'est encore nécessaire. Pour le nouvel état E9, on calcul les configurations candidates sur la base des configurations candidates de l'état précédent. Les couples de noeuds `(ti2),(p1)` et `(ti5),(p1)` peuvent également être candidats. Ils forment respectivement les noeuds `(ta1)` et `(ta2)` qui sont stockés dans les configurations candidates (voir figure V.12).

```
A10. GOTO C-B P_C-B_CG_CP_C P_C-B_CG
```

Signification. La composition chariot-boîte doit aller dans une position à propriété de généralité.

Validation. Aucun mouvement n'est nécessaire à calculer, le noeud (ta0) est correct. Dans le nouvel état E10 les configurations candidates pour le chariot sont la composante connexe de (ta0), de (ta1) et de (ta2). Les composantes connexes de (ta0) et de (ta1) sont identiques : elles correspondent à la même prise. La composante connexe de (ta2) par contre est différente. On peut en voir une illustration sur la figure V.12.

A11. GOTO C-B P_C-B_CG P_C-B_CG_CP_C

Signification. La composition chariot-boîte doit aller dans une position à propriété de lien.

Validation. Le noeud (ta0) est correct, pas de mouvement à calculer. Un nouvel état E11 est créé (voir figure V.12). Les configurations candidates sont propagées.

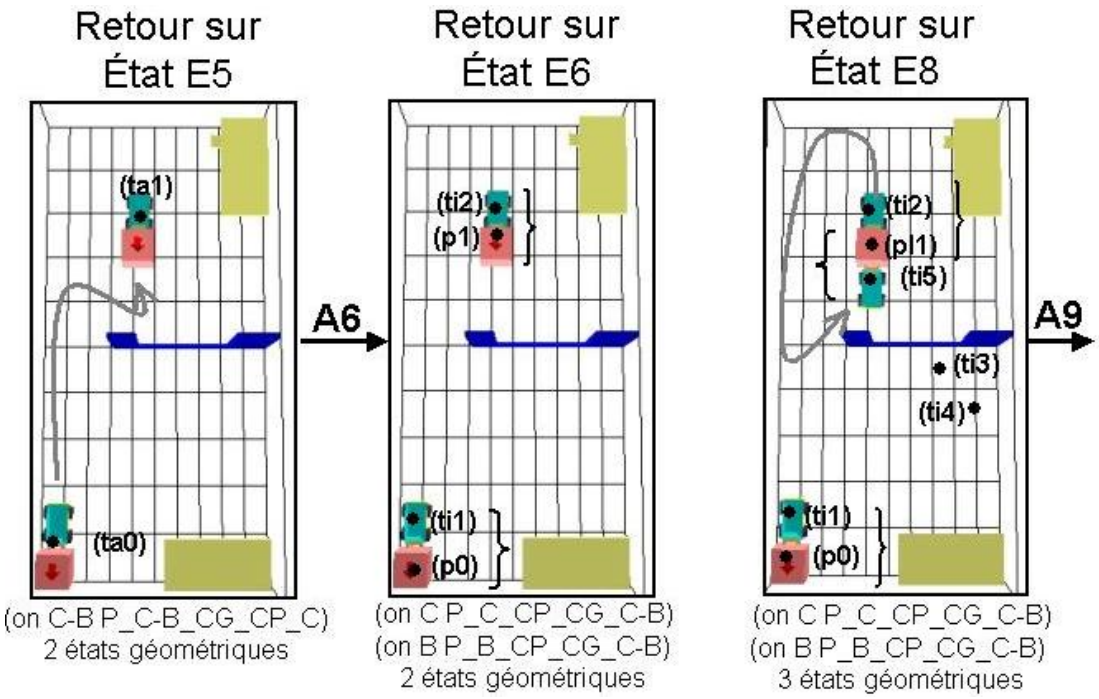

FIG. V.13 – Validation : L'action A12 n'est pas valide avec les états géométriques courants, on revient sur l'état E5, E6 puis E8.

La douzième action proposée est

A12. UNGRASP C-B P_C-B_CG_CP_C

```
C   P_C_CP_CG_C-B
B   P_B_CP_GOAL
```

Signification. La composition chariot-boîte doit se décomposer de manière à ce que la boîte se retrouve dans sa configuration finale.

Validation. L'unique état géométrique de E11 ne permet pas de réaliser cette action. Par contre dans les candidats géométriques de E11, un noeud peut être candidat, le noeud (ta3). Remarquez que si aucun noeud n'avait été candidat, l'action n'aurait même pas pu être proposée. Elle aurait été filtrée dés la première étape de l'heuristique principale (cf. V.2.4).

Avec ce noeud candidat, le planificateur va remonter toute la chaîne de mouvement à calculer nécessaire pour que le noeud (ta3) fasse partie de l'état géométrique de E11. Le planificateur va faire le raisonnement suivant : pour que (ta3) fasse partie des états géométriques de E11, il aurait fallu que :

– (ta2) fasse partie de l'état géométrique de E9.
– Le couple (ti5),(p1) fasse partie de l'état géométrique de E8.
– Le couple (ti2),(p1) fasse partie de l'état géométrique de E6.
– (ta1) fasse partie de l'état géométrique de E5.

En exploitant ce raisonnement, qui peut être vu comme un retour arrière, le planificateur va déduire les mouvements à valider. Ce sont les mouvements allant de (voir les figures V.13 et V.14) :

– (ta0) à (ta1), ce qui provoque la création d'un nouvel état géométrique pour l'état E5 puis par propagation pour l'état E6 et E7.
– (ti2) à (ti5), et donc la création d'un nouvel état géométrique pour l'état E8 et par propagation pour l'état E9 et E10.
– (ta2) à (ta3), et donc la création d'un nouvel état géométrique pour l'état E11.

C'est lors de ces validations de mouvement que le planificateur s'assurera que les chemins ne rentrent pas en collision avec les robots et les objets. Tous ces mouvements ont pu être validés.

L'action A12 est maintenant valide est produit la création d'un état E12 valide et satisfaisant le but (voir figure V.14. Un plan a été trouvé !

Conclusion sur la validation

L'avantage de notre processus de validation, c'est qu'il ne calcule la validité des mouvements que lorsque c'est absolument nécessaire. Autrement dit, à chaque nouvelle action du plan symbolique, de nouvelles contraintes vont apparaître. Elles ne seront prises en compte qu'à ce moment précis.

Dans l'exemple précédant, la contrainte de placement de la boîte n'apparaît qu'à la fin de l'exploration géométrique du plan symbolique. Ce n'est donc qu'à la fin de

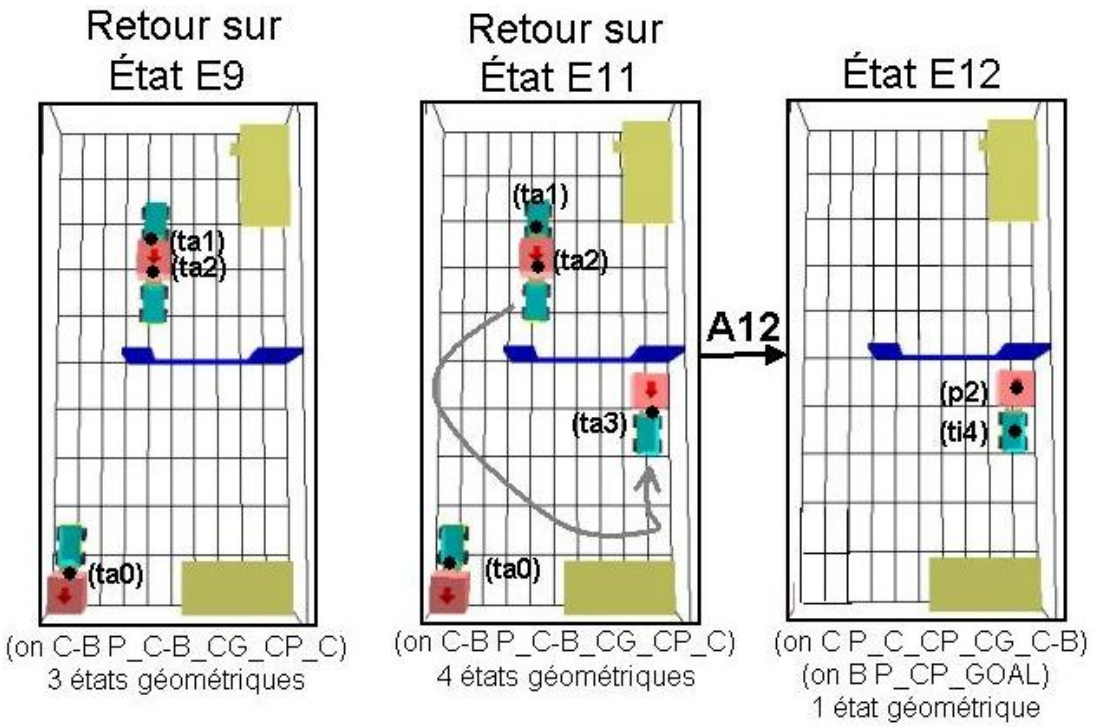

FIG. V.14 – Validation : suite du retour arrière permettant de valider l'action A12.

l'exploration du plan symbolique que l'on validera les mouvements nécessaires à la satisfaction de cette contrainte. Le processus peut donc être très économe en tests de collisions dans le cas de problèmes simples.

La validation reste un processus coûteux qui peut être dans le pire des cas exponentiel en nombre de robots. Ce processus est toutefois complet : il assure de trouver une solution si celle-ci existe au sein des roadmaps. Nous avons limité le nombre d'appel au détecteur de collisions en utilisant de la mémoire. Nous stockons dans une structure de type arbre binaire les résultats de la validité des arêtes pour un contexte géométrique précis. Ce mécanisme nous permet une nette amélioration des performances : le temps passé dans le processus de validation peut être divisé par 100 sur certains problèmes.

Dans l'implémentation courante d'aSyMov, nous ne calculons jamais de nouveaux états géométriques pour les positions à propriété générale. Ce calcul nous permettrait pourtant de trouver des positions intermédiaires aux robots obstruant des chemins indispensables à la réussite de la planification. C'est la raison même de la notion de position à propriété de généralité. Malheureusement, la combinatoire serait trop importante.

Nous préférons savoir qu'un certain nombre de problèmes ne sont pas résolubles avec aSyMov : ce sont les problèmes où un robot est gênant et où aucun des plans possibles ne suggère de le déplacer pour qu'il aille accomplir une tâche, c'est à dire que le robot n'a aucune utilité symbolique. Autrement dit, le problème est mal défini initialement. Ici, nous nous rapprocherions d'un problème de planification de mouvements coordonnés bien connu. Ce n'est pas celui qui nous intéresse de prime abord.

V.2.6 Expansion de roadmap

Principes

Un des principes fondateur de aSyMov est la découverte de la topologie de *CS* en cours de planification. C'est ce principe qui rend le front de recherche si difficile à gérer. Pour l'heure, nous pensons que nous ne pouvons pas trouver de plan efficacement sans lui. En effet :

– Les ajouts des noeuds peuvent ainsi être réalisés suivant les besoins réels du problème. Inutile en effet d'ajouter un nombre de noeuds trop important aux roadmaps qui ne seront pas ou peu utilisées. En ce sens notre représentation symbolique nous guide dans ces ajouts de noeuds.

– Les ajouts de noeuds pourront se faire en prenant en compte le contexte géométrique probable. Inutile en effet d'ajouter des noeuds en collision avec les autres robots ou objets au moment où on voudrait s'en servir pour obtenir un chemin. Plus précisément, lorsque nous ajoutons de nouveaux noeuds, nous sélectionnons aléatoirement un état géométrique de l'état d'aSyMov courant et nous le prenons comme contexte de collision.

Ces deux arguments seront illustrés empiriquement dans le chapitre évaluant le système (cf. VII.1.2).

Nous rappelons que lors de l'extension d'un état, le planificateur peut faire appel à deux types d'actions d'expansion de roadmap : 1 action d'expansion de roadmaps en largeur ou *Facteur de Branchement* action d'expansion en profondeur.

Les règles d'enrichissement

A chaque sélection d'une action d'expansion de roadmaps, le planificateur va tenter d'ajouter un certain nombre de noeuds aux roadmaps. Ces ajouts sont réalisés au travers d'un jeu de *règles d'enrichissement*. Un développeur de domaine pour aSyMov s'attachera à créer une règle d'enrichissement pour chaque sous-espaces représentés symboliquement. Ainsi chaque roadmap aura sa régle d'enrichissement associée.

Une règle d'enrichissement comporte deux parties : des coûts et des effets. Pour chaque sélection d'une action d'expansion de roadmaps, le planificateur jugera grâce

aux coûts la règle dont le planificateur devra appliquer les effets. Ceci sera fait un nombre paramétrable de fois (quatre dans la version courante d'aSyMov).

Les **coûts d'une règle d'enrichissement** sont basés sur :

– La taille des roadmaps : le coût de la règle dépendra du nombre de noeuds de la roadmap ou d'une fonction paramétrable linéaire ou quadratique du nombre de noeuds de cette roadmaps.

– Le nombre de noeuds sources non connectés : un coût fonction du nombre de noeud source d'une roadmap relative non encore lié à un noeud d'échappement. Ce coût permet d'évaluer l'intérêt à étendre une roadmap relative. Dans l'implémentation courante, les roadmaps relatives sont également construites en cours de recherche. Ceci sera discuté en conclusion de cette thèse (cf. chapitre VIII)

– L'intérêt symbolique : le coût de la règle dépendra de l'intérêt de l'expansion d'une roadmap indiqué par les plans possibles courants. Ces coûts sont différemment distribués suivant que l'action d'expansion est une action d'expansion en largeur ou en profondeur. Pour l'unique action d'expansion de roadmap en largeur, les coûts d'intérêt symbolique sont distribués uniformément entre toutes les règles liées aux roadmaps mises en jeu par les actions applicables partant de l'état courant. Pour les actions d'expansion en profondeur, les coûts sont distribués entre toutes les règles liées aux roadmaps mises en jeu par le plan possible partant d'une action applicable précise. Plus l'action est profonde dans le plan moins nous souhaitons ajouter de noeud aux roadmaps que l'action met en jeu, plus les coûts d'intérêt symbolique des règles liées à ces roadmaps seront important.

Ces coûts basés sur l'intérêt symbolique nous permettent de dire que les choix sur l'expansion des roadmaps sont indépendants du domaine. En effet, ces coûts vont se paramétrer automatiquement selon le problème résolu.

Les **effets d'une règle d'enrichissement** comportent :

– Un identificateur.

– La roadmap à étendre.

– Les liens d'héritage qui vont être automatiquement activés. Par exemple, nous pouvons décider que pour une règle concernant l'expansion d'une roadmap de transfert, l'ajout d'un noeud dans cette roadmap ajoutera automatiquement un noeud dans la roadmap de transit liée.

– Les liens d'héritage qui doivent être utilisés pour la création de noeud. Par exemple, nous pouvons décider qu'une règle concernant l'expansion d'une roadmap de transfert, l'ajout d'un noeud dans cette roadmap ne pourra se faire qu'à partir d'un noeud de la roadmap de placement.

– Le nombre de noeuds à ajouter à l'activation de la règle.

– La méthode de construction de la roadmap : RRT, PRM classique, PRM avec visibilité, recherche près d'un noeud...

Les effets d'une règle peuvent également faire appel à un ensemble de sous règles. Un développeur de domaine pour aSyMov s'attachera à créer une règle pour toutes les roadmaps représentées symboliquement. Ainsi, les coûts liés aux intérêts symboliques pourront être correctement distribués. Dans le fichier "lien" (cf. V.2.1), l'identificateur de la règle d'enrichissement sera alors relié au symbole du sous-espace.

Un certain nombre de roadmaps ne sont pas représentées au niveau symbolique. C'est le cas notamment des roadmaps relatives. La règle d'enrichissement de la roadmap de transit appellera un jeu de plusieurs sous règles : une pour l'enrichissement de la roadmap de transit proprement dite et les autre pour l'enrichissement des roadmaps relatives pour les différents objets. Les coûts distribués entre les deux sous règles seront paramétrés à la main par le développeur.

Ce dernier point n'est pas satisfaisant mais notre expérience de ce paramétrage n'est pas encore suffisante pour l'automatiser. Toutefois, ce paramétrage n'est réalisé qu'une seule fois, il est ensuite utilisé tel quel pour toutes les instances du problème. Le paramétrage ne semble pas devoir être modifié lorsque nous modifions un problème.

Sélection des règles d'enrichissement

Lors de la sélection d'une action d'expansion de roadmap, le planificateur calcule tout d'abord les coûts de chacune des règles d'enrichissement à sa disposition. Il procède ensuite à une sélection itérative de règles par tirage aléatoire pondéré par le coût des règles.

Un exemple

Pour le problème du chariot et de la boîte (V.8), nous avons défini deux règles d'enrichissement principales : "Transit C" et "Transfert C-B". Elles ont toutes les deux un coût lié à leurs intérêts symboliques. Elles font toutes deux appel à un ensemble de sous règles.

Pour "Transit C", nous avons défini deux sous règles : "Transit" et "Relative boîte". Les effets de "Transit" sont d'ajouter deux noeuds selon une méthode PRM classique dans la roadmap de transit du chariot `C`. Les effets de "Relative boîte" sont d'ajouter deux noeuds selon une méthode RRT dans la roadmap relative du chariot `C` par rapport à la boîte `B`. Les coûts entre ces deux sous règles dépendent du nombre de noeuds de chacune de ces roadmaps. Nous voulons qu'il y ait quatre fois plus de noeuds dans la roadmap de transit que de noeuds dans la roadmap relative. Nous y ajoutons un coût basé sur le nombre de noeuds sources non connectés.

Pour "Transfert C-B", nous avons défini trois sous règles : "Nouveaux placements", "Grasp inter Placement" et "Transfert" :

– “Nouveaux placements” ajoutera un noeud dans la roadmap de placement de la boîte B. Cette règle nous permettra éventuellement de trouver des nouveaux placements pour poser la boîte dans une position intermédiaire.

– “Grasp inter Placement” créé deux noeuds dans la roadmap de transfert de la composition C−B. Elle construit ces noeuds par cinématique inverse à partir des noeuds de la roadmap de placement de la boîte. Cette sous règle est très pratique pour créer les noeuds de transfert correspondant aux placements initiaux, finaux ou intermédiaires de la boîte. Cette règle produit en fait ce qu’on avait appelé précédemment des noeuds sources (cf. IV.2.2). Des liens d’héritage sont alors automatiquement activés lors des ajouts de noeuds grâce à cette règle. Elle ajoutera des noeuds par héritage dans la roadmap de transit et des noeuds, après le changement de repère adéquat, dans la roadmap relative.

– “Transfert” crée deux noeuds dans la roadmap de transfert sans contrainte particulière. Cette règle permettra de relier les noeuds créés par “Grasp inter Placement”. Aucun lien d’héritage n’est activé.

Les coûts entre ces trois sous règles sont paramétrés en fonction du nombre de noeuds qu’elles créent.

Chapitre VI

Extraction et améliorations du plan

Un état final a été atteint. C'est la garantie qu'un plan solution existe. Dans ce chapitre nous allons détailler comment se présente le plan solution. Nous montrons ensuite comment nous procédons à des améliorations du plan séquentiel pour le rendre plus court en nombre d'action et en longueur de chemin parcouru. Dans un second temps, nous montrons comment nous pouvons passer d'un plan séquentiel à un plan parallèle. Une fois encore ce chapitre montre une intégration entre des techniques de planification de tâche et des techniques de planification de mouvement.

VI.1 Amélioration du plan séquentiel

Un plan solution d'aSyMov séquentiel se définit par la suite S de quadruplet :
$S = ((Es_0, Eg_0, A_0, SCL_0), ..., (Es_n, Eg_n, A_n, SCL_n))$
où :
– Es_i est un état symbolique
– Eg_i est un état géométrique associé à Es_i nous ayant permis d'atteindre un état d'aSyMov but. C'est une configuration de CS_{free}.
– A_i $(0 <= i < n)$ est une action symbolique permettant d'atteindre l'état symbolique Es_{i+1}.

– SCL_i est une suites de chemins locaux (i.e. le chemin) permettant d'atteindre l'état géométrique Es_{i+1}. Cette suite peut être nulle dans le cas des actions "purement symbolique".

VI.1.1 Améliorations symboliques

Comme cela est signalé dans les chapitres précédents, nous avons volontairement autorisé le planificateur à visiter plusieurs fois le même état symbolique pour éventuellement faire des opérations de changement de prise au cours de manipulation. Maintenant qu'une solution existe, nous pouvons analyser le plan S pour éliminer d'éventuelles boucles inutiles.

La **première étape** consiste à éliminer dans S tous les quadruplets numéroté de i à $j-1$ tel que $Es_i = Es_j$ et $Eg_i = Eg_j$. En d'autre terme il y avait dans le plan deux états strictement identiques du point de vue géométrique comme du point de vue symbolique. Nous éliminons alors la branche du plan comprise entre ces deux états.

La **seconde étape** est une analyse plus fine de S. Les positions ayant une propriété de zone ou de lien sont peut-être utilisées dans le plan comme des positions à propriété de généralité : le plan n'utilise pas leurs propriétés pour atteindre l'état but. Nous procédons en suivant l'évolution des positions sur le plan symbolique. Si au cours du plan, aucune action A_i n'utilise la propriété de zone ou de lien d'une position, celle-ci est alors remplacée par sa position à propriété de généralité associée. Une fois ces remplacements effectués, on réitère la première étape.

Pour savoir si une action A_i "utilise" une propriété, il suffit de vérifier si celle-ci possède dans ses préconditions un fait "`has_property`" concernant la position pour une position à propriété de zone ou un fait `link` concernant la position pour les positions à propriété de lien. L'ensemble des positions P_{unuse} dont on n'utilise pas la propriété est :

$$P_{unuse} = \{p \in P / \exists pr \in Pr / (has\_property\ p\ pr)\ and\ \neg(\exists i \in [1;n] / precond^{+}(A_i) \cap (has\_property\ p\ pr) \neq \emptyset\} \cup \{p \in P / \exists p_1 \in P / (link\ p\ p_1)\ and\ \neg(\exists i \in [1;n] / precond^{+}(A_i) \cap (link\ p\ p_1) \neq \emptyset\}$$

La position à propriété de généralité associée à une position à propriété de zone ou de lien est la position n'ayant pas de fait `link` ou `has_property` la concernant et elle est rattachée au même symbole de sous-espace que la position à propriété de zone ou de lien à travers les faits `belongs-to`.

La **troisième étape** cherche à réduire encore d'éventuelles redondances dans S. L'analyse consiste à rechercher les quadruplets où Es_i est égal à Es_j $(j > i)$ mais où Eg_i est différent de Eg_j. Autrement dit, le plan passe deux fois par le même état symbo-

lique mais avec deux états géométriques différents. Cette situation est assez fréquente, surtout après la deuxième étape. Quand cette situation est remarquée, le planificateur tentera de remplacer à partir du j-ème quadruplet l'état géométrique Eg_j par Eg_i. S'il parvient à atteindre de nouveau l'état but, on pourra supprimer tous les quadruplets de S compris entre $i+1$ et j. Nous venons de supprimer des parties de déplacement inutiles. Si ce n'était pas le cas, ces mouvements étaient certainement utiles pour trouver une nouvelle prise par exemple.

On peut se demander si les trois étapes d'analyse faites ci-dessus ne peuvent pas être intégrées dans le contrôle de la recherche. Nous ne pensons pas que ce soit atteignable. En effet les redondances d'états symboliques et/ou géométriques peuvent mener le planificateur au succès, mais en cours de recherche nous ne pouvons pas juger du bien-fondé de telles répétitions, elles ne peuvent être jugées qu'à la réussite de la planification. Dans notre exemple pour expliquer le processus de validation (cf. V.2.5), en cours de recherche , c'est à dire avant le retour arrière survenant à la validation de l'action `A12` on a :

$Es_3 = Es_5$ et $Eg_3 = Eg_5$

Les états $E3$ et $E5$ ont la même partie symbolique et le même état géométrique. Pourtant, ces deux états sont absolument nécessaires pour le plan final. Cela est dû à leurs différences au niveau des configurations candidates. Les boucles mettant en jeu des états strictement identiques, y compris au niveau des configurations candidates, peuvent être éliminer en cours de recherche.

Dans nos expérimentations, nous parvenons à réduire fortement le nombre de boucles inutiles dans les plans. Par exemple, dans les problèmes de type "chariots élévateurs", il n'est pas rare de voir dans les plans avant les nettoyages les robots tenter de saisir les boîtes suivant plusieurs prises pour finalement revenir à la prise qu'ils avaient choisie au début avant de transférer la boîte. C'est typiquement le genre de situations évitées ici.

VI.1.2 Améliorations géométriques

Le plan est désormais plus réduit, nous pouvons désormais travailler sur les composantes SCL_i de notre plan.

L'algorithme de validation est un algorithme paresseux. Il s'arrête à la première solution trouvée. La première amélioration que nous appliquons sera de trouver non pas la première mais la meilleure suite SCL_i pour chacun des quadruplets de la suite S représentant le plan solution. Pour cela nous faisons appel à un algorithme A* qui nous garantira le chemin le plus court dans les roadmaps. Nous avons la guarantie qu'une

solution existe. Cette étape peut être assez longue, elle peut donc être facultative.

La seconde étape consiste en une optimisation des chemins ainsi obtenue. Pour cela nous appliquons les techniques d'optimisations introduites notamment dans [Laumond 94]. Cela consiste à prendre deux points aléatoirement sur le chemin puis d'utiliser ces points pour réduire la longueur du chemin (figure VI.1).

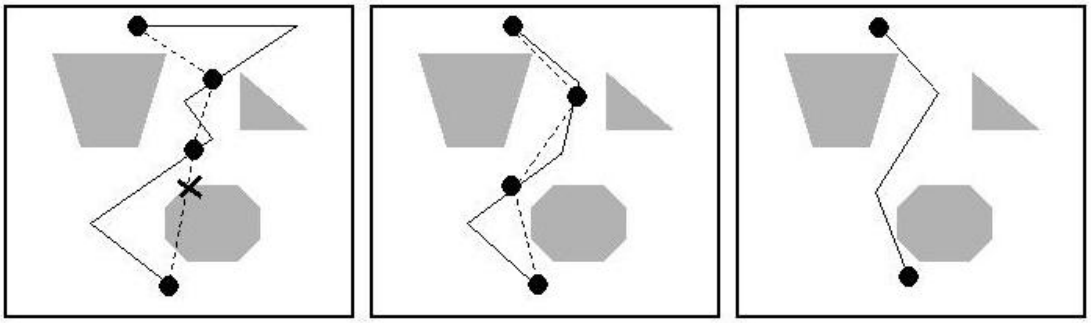

FIG. VI.1 – L'algorithme d'optimisation de chemin par tirage aléatoire.

Nous pouvons également utiliser des techniques dites de "bandes élastique" [Brock 97]. Le principe est d'appliquer des forces tendant à raccourcir la trajectoire comme un élastique et des forces repoussant la trajectoire des obstacles. Cette méthode contrairement à la précédente n'est pas généralisable à toutes les méthodes locales.

VI.2 Parallélisation du plan

Après l'élimination des boucles inutiles et les améliorations des chemins, il reste un inconvénient majeur : pour les plans mettant en jeu plusieurs robots, la séquentialité du plan est clairement sous optimale. Cette section s'applique à proposer une parallélisation des plans d'aSyMov. Pour cela nous allons procéder en deux étapes.

Premièrement, grâce à une technique issue du domaine de la planification de tâches, nous parallélisons le plan symbolique séquentiel. Le résultat de cette étape se présentera comme un ordre partiel des actions A_i calculé sur la base de la seule représentation symbolique.

Deuxièmement, grâce à une extension de méthode issue de la planification de mouvement, nous synchroniserons les trajectoires qui peuvent avoir lieu en même temps.

VI.2.1 Parallélisation du plan symbolique

Techniques de l'état de l'art

La technique de recherche de plan dans l'espace d'état présente l'inconvénient majeur de donner un ordre total entre les actions. La recherche dans l'espace des plans partiels produit un plan partiellement ordonné voir partiellement instancié. Cette seconde technique retourne des plans plus flexibles éventuellement non totalement instancié et donc de meilleure "qualité". La recherche dans l'espace d'état, par contre, est nettement plus efficace en terme de temps de calcul car elle peut s'appuyer sur des heuristiques puissantes comme des relaxations "à la graph-plan" [Blum 97] .

Certains chercheurs ont donc eu l'idée de rendre partiel l'ordre total produit par un plan séquentiel, et ceci en post-traitement. La première approche a été proposée dans [Régnier 91]. Elle consiste à supprimer les relations d'ordre qui ne sont pas utiles entre les actions. Ces relations proviennent en fait de la méthode de recherche et non pas des contraintes du problème.

Backstrom propose lui dans [Backstrom 98] d'approcher le problème en désordonnant le plan en éliminant des relations d'ordre ou en réordonnant le plan en modifiant des relations d'ordre. Le but étant de produire un plan moins contraint afin de minimiser le temps de son exécution. En effet des tâches pourront être exécuter en parallèle. Il montre aussi que trouver une solution optimale en temps à ce problème est NP-hard. Do et Kambhampati [Do 03] généralisent ceci pour des plans issus de planificateur gérant des attributs numériques et temporels. Ils proposent une solution au problème de "partialisation" en l'encodant dans un C.S.P. Dans des travaux plus récents [Pecora 04], est proposée une séparation encore plus claire entre planification et ordonnancement.

Vu que nous nous situons dans une planification de tâches non temporelle, nous pouvons nous satisfaire de la méthode proposée dans [Régnier 91]. Elle est brièvement expliquée ci-après.

Aperçu de la méthode basée sur la table triangulaire

La table triangulaire est un tableau représentant les dépendances entre les actions. Sa construction est définie dans [Fikes 72]. Pour illustrer le propos nous choisissons une nouvelle fois le problème interrupteur/radio.

L'état initial symbolique du problème est :

```
(on FORKLIFT P_FORKLIFT_CP_INIT)
(on ARMBOT P_ARMBOT_CP_INIT)
(on CBOX P_CBOX_CP_INIT)
```

Le but est :

```
(on F_CBOX P_F_CBOX_CG_GOAL)
```

```
(data-received)
```

La composante symbolique du plan qu'a trouvée aSyMov est (dans la nomenclature choisie en III.2.3 :

```
1  - GOTO ARMBOT P_ARMBOT_CP_INIT P_ARMBOT_CP
2  - GOTO FORKLIFT P_FORKLIFT_CP_INIT P_FORKLIFT_CP
3  - GOTO FORKLIFT P_FORKLIFT_CP P_FORKLIFT_CP_CG_F_CBOX
4  - GRASP FORKLIFT P_FORKLIFT_CP_CG_F_CBOX
           CBOX     P_CBOX_CP_INIT
           F_CBOX   P_F_CBOX_CG_CP_FORKLIFT
5  - GOTO ARMBOT P_ARMBOT_TI P_ARMBOT_CP_SWITCH
6  - SWITCH ARMBOT P_ARMBOT_CP_SWITCH
7  - GOTO F_CBOX P_F_CBOX_CG_CP_FORKLIFT P_F_CBOX_CG
8  - GOTO F_CBOX P_F_CBOX_CG P_F_CBOX_CG_GOOD_RECEPTION
9  - RECEIVE-DATA F_CBOX P_F_CBOX_CG_GOOD_RECEPTION
10 - GOTO F_CBOX P_F_CBOX_CG_GOOD_RECEPTION P_F_CBOX_CG
11 - GOTO F_CBOX P_F_CBOX_CG P_F_CBOX_CG_GOAL
```

La table triangulaire met en relation les faits de l'état initial puis ceux rajoutés par l'application des actions avec les actions qui les utilisent. Après une analyse de la table, on peut extraire un ordre partiel des actions. Ce qui nous donne pour notre exemple le plan symbolique parallèle de la figure VI.2.

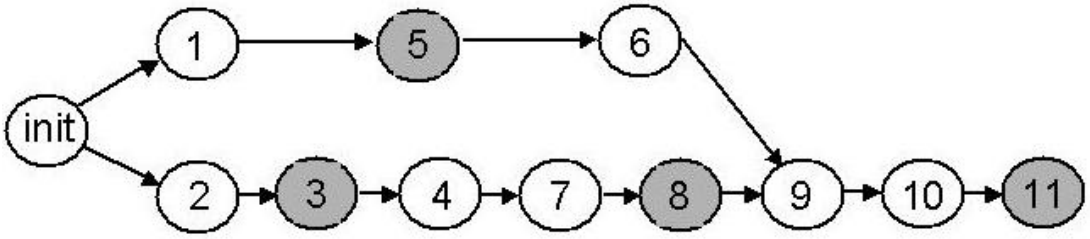

FIG. VI.2 – Plan parallèle obtenu sur la composante symbolique du plan trouvé par aSyMov pour le problème interrupteur/radio.

Nos plans concernent des robots qui généralement n'ont pas le don d'ubiquité. Cela a pour conséquences de créer un ordre partiel entre les actions qui s'organise en branches : une branche par robot. Cette propriété va être utilisée par la suite. Le problème que nous avons maintenant est de savoir comment synchroniser ces branches d'actions d'un point de vue géométrique. En effet, certaines de ces actions symboliques sont reliées à des chemins (actions grisées sur la figure VI.2) qu'il va falloir synchroniser. C'est ce qui est montré ci-après.

VI.2.2 Parallélisation du plan aSyMov

Cette partie à été réalisée avec Akin Sisbot dans le cadre de son stage en D.E.A. [E.A.Sisbot 04].

Spécification du problème

Le problème que nous souhaitons résoudre ici peut être décrit comme une série d'entrées sorties. Les entrées du problème sont :

– Le plan symbolique parallèle trouvé en VI.2.1.

– Les chemins (optimisée en VI.1.2) que les robots doivent emprunter pour accomplir certaines actions.

– Les contraintes dynamiques des systèmes. Nous ne prendrons en compte que la vitesse maximale des robots. Nous verrons que cela n'est pas entièrement satisfaisant.

En sortie nous désirons obtenir une solution optimale en temps à l'exécution du plan. Pour cela on ne jouera que sur le profil des vitesses des robots. On supposera que les actions "symboliques" sont instantanées.

Techniques de l'état de l'art

La planification de mouvements est connue pour être exponentielle par rapport au nombre de degrés de liberté du système considéré. Lorsqu'on veut planifier pour plusieurs systèmes physiquement indépendants, il peut être avantageux de planifier les mouvements nécessaires pour chacun des systèmes indépendamment puis d'utiliser les résultats obtenus pour trouver une solution au problème général. La complexité du problème est alors réduite. Quelques recherches ont donc été naturellement réalisées sur ce sujet. Les avancées récentes sur la planification de mouvements en environnement dynamique travaillent également sur un sujet assez connexe. La littérature est donc assez riche. Nous ne citerons que quelques exemples. Par rapport à cet état de l'art, notre problème possède trois spécificités :

– Au cours d'un plan, les robots saisissent ou relâchent des objets. Leurs formes géométriques évoluent donc au cours du temps.

– Nous avons un ordre partiel à respecter entre les actions et donc entre les chemins. Nous aurons donc un jeu de contraintes supplémentaires.

– Nous savons qu'une solution existe : c'est le plan séquentiel.

L'approche des diagrammes de task-completion [O'Donnell 89] travaille sur un problème proche du notre. Les trajectoires sont calculées de manière indépendante pour chacun des deux robots. Les trajectoires sont des séquences de segments. On construit un diagramme de task-completion qui est une grille où les segments de la

première trajectoire sont représentés en abscisse et les segments de la seconde trajectoire sont représentés sur l'axe des ordonnées. Une case de la grille est "noircie" si les segments correspondants sont en collision. Le problème est ensuite un problème d'ordonnancement entre les segments, ce qui revient à trouver un chemin dans la grille qui ne traverse pas de cases noires. La grille peut également être construite en prenant comme segments des temps d'exécution de trajectoire. Cette approche est initialement prévue pour deux robots mais son application pour plusieurs robots a donné de bons résultats [Siméon 02]. La technique que nous allons proposer est clairement le prolongement de ces travaux.

Une solution à une problématique proche de la notre est également donnée dans [Akella 02]. Contrairement à nous, Akella propose une résolution où les trajectoires des robots sont connues non seulement par leurs chemins mais aussi par leurs profils de vitesse. La résolution du problème ne joue alors que sur le top de départ pour chaque trajectoire. Le problème est ensuite traité comme un problème d'optimisation. Une solution optimale à ce problème est extraite. Malgré tout, l'hypothèse des vitesses connue nous semble trop restrictive et produit des solutions peu optimales.

Ceci a été amélioré dans [Peng 03] où cette fois la résolution tente de produire le profil de vitesse. Les différentes contraintes sur les limites de vitesses, d'accélération voir des contraintes dynamiques plus fines et les zones de collision sont formulés dans un même problème. Le problème tel qu'il est formulé n'a pas de méthode de résolution optimale. Peng propose alors de résoudre deux problèmes relaxés qui lui donneront les bornes inférieure et supérieure de la solution optimale. Si les solutions donnent le même résultat, alors on a trouvé la solution optimale.

On peut également citer le paradigme d'insertion de plan ([Qutub 98]) plus adapté à un ensemble de robots réellement indépendants alors que nous sommes dans un cadre de planification centralisé. Lorsqu'un robot reçoit un nouveau but, il crée un plan individuel. Avant son exécution le robot doit garantir la validité du plan dans le contexte multi-robots. L'adaptation de cette solution à notre problématique nécessiterait une formulation différente de notre problème. De plus, nous avons déjà trouvé un plan coordonné (i.e. le plan séquentiel), nous allons plutôt chercher à l'améliorer.

Les méthodes qui déforment les trajectoires, comme celle de la bande élastique [Brock 02], si situe à une autre niveau : celui de l'exécution de trajectoires.

Notre méthode implémentée

Nous présentons ici notre méthode pour résoudre le problème que nous avons spécifié. Nous la présentons sous la forme d'un algorithme détaillé point par point. On peut noter dès maintenant que la méthode est adaptée à un nombre quelconque de robot. La méthode sera discutée dans un deuxième temps.

1. Identification des robots de base : Nous voulons profiter de la propriété du plan parallèle qui s'organise en branches, une par robot. Or nous considérons depuis le début de ce manuscrit qu'un robot se composant avec un objet forme un nouveau robot. Nous identifions ici les robots les plus "simples" évoluant dans le plan sans prendre en compte les objets. Nous les nommons *robot de base*. Dans notre exemple courant, nous avons quatre robots : `FORKLIFT`, `ARMBOT`, `CBOX` et `F_CBOX`. Les deux robots de base sont `FORKLIFT` et `ARMBOT`. Cette identification se fait grâce à une analyse des instances des prédicats `compose` du problème symbolique (III.2.1). Nous rappelons que les prédicats `compose` sont statiques. l'ensemble des robots de base R_base est un sous ensemble de l'ensemble des robot R suivant :

$$R_{base} = \{r \in R / \forall r_1 \in R, \forall r_2 \in R \; not(compose\ r\ r_1\ r_2)\}$$

2. Construction des axes : Pour chaque robot de base, nous construisons un axe qui représentera les actions concernant le robot de base. Les actions symboliques seront représentées par un point sur cet axe, les actions attachées à un chemins seront représentées par un segment de longueur proportionnelle à la longueur du chemin. Les segments liés à une trajectoire sont ensuite découpés en plusieurs petits segments qui représentent une discrétisation du chemin. Chacun des points créés représentera alors une configuration du robot sur ce chemin. Le pas de discrétisation est donné par l'utilisateur. Par défaut il est égal à la moitié de la plus petite dimension de la boîte englobante du robot[1]. Un ajustement automatique du pas de discrétisation est toutefois opéré pour que deux configurations soient espacée de la même valeur.

3. Initialisation de la grille : On crée une grille de dimension égale au nombre de robots de base présent dans le plan. La grille a pour axes ceux calculés précédemment. Une "case" de la grille représente donc une configuration de chaque robot de base sur une de ses trajectoires. La figure VI.3 représente la grille construite pour notre exemple sur le problème interrupteur/radio.

4. Représentation des contraintes d'ordre : La plupart des contraintes d'ordre sont déjà représentées sur la grille. Elles correspondent aux contraintes d'ordre des "branches" du plan parallèle. Il reste à représenter les contraintes d'ordre entre les actions concernant des robots de base différents. Cela revient à interdire des cases de la grille. Dans notre exemple, l'action 9 doit obligatoirement avoir lieu après l'action 6. Cela revient interdire un certain nombre de cases sur la grille. Ceci est illustré sur la figure VI.3.

5. Remplir la grille : Pour les cases non encore "noircies", nous faisons appel au détecteur de collisions pour savoir si les configurations de chaque robot de base corres-

[1] Les boîtes englobantes sont calculés par Move3d en prétraitements pour améliorer les performances du détecteur de collision

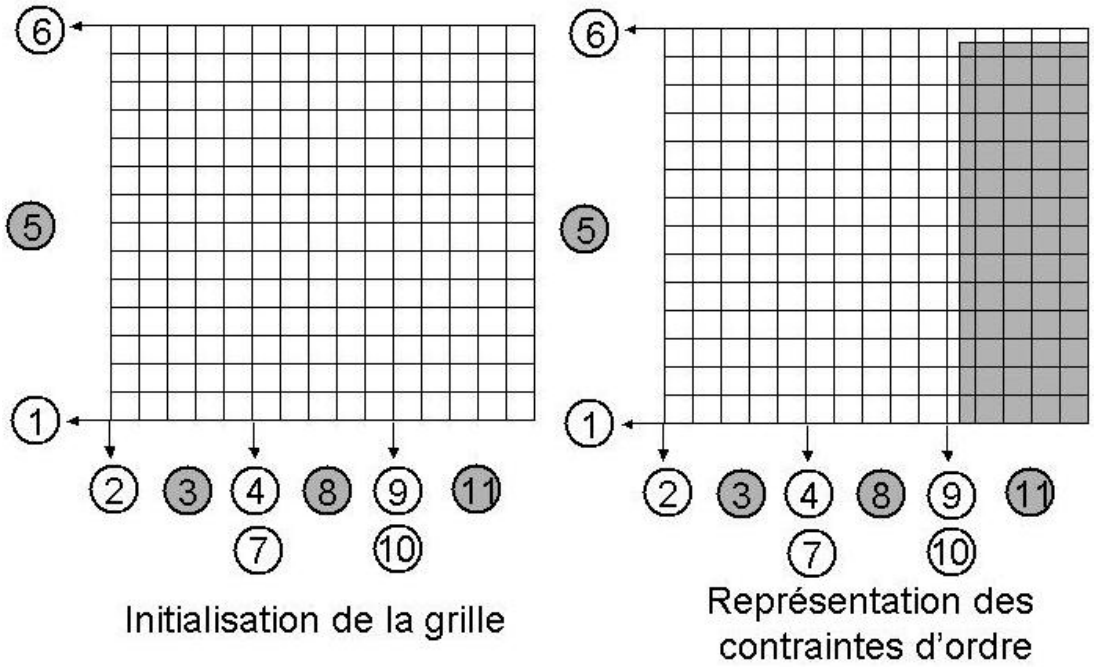

FIG. VI.3 – Initialisation de la grille et représentation des contraintes d'ordre.

pondant à la case sont en collision. Cela nous crée des "obstacles" représentant le fait que les robots ne puissent pas être sur ces configurations respectives en même temps. La figure VI.4 illustre ceci. Nous pouvons remarquer que dans le cas où nous aurions plus de deux robots de base, nous ne faisons les tests de collisions que deux à deux. Les résultats sont ensuite projetés dans la grille. L'algorithme reste en $O(((nb\ case)^2 * (nb\ robot)^2)/2)$.

6. Fermeture sud-ouest des obstacles : Nous ne voulons pas que les robots reculent sur leurs trajectoires. Nous chercherons un chemin ne rentrant pas en collision avec les "obstacles" de la grille composé uniquement de segments croissants. Certaines zones de la grille sont donc inaccessibles. Nous opérons une fermeture dite "sud-ouest" des obstacles illustrée sur la figure VI.4

7. Trouver une solution de synchronisation : Notre grille est maintenant complète. Nous décidons de trouver le chemin le plus court menant à l'angle nord-est de la grille. Pour cela nous utilisons un algorithme A* sur le graphe de visibilité issu de la grille. Pour ce graphe nous ne conservons que les points les plus proches des obstacles. Nous trouvons donc les configurations par lesquelles les robots doivent arriver en même temps. Nous avons implémenté ensuite une solution dans ce qui est souvent appelé le

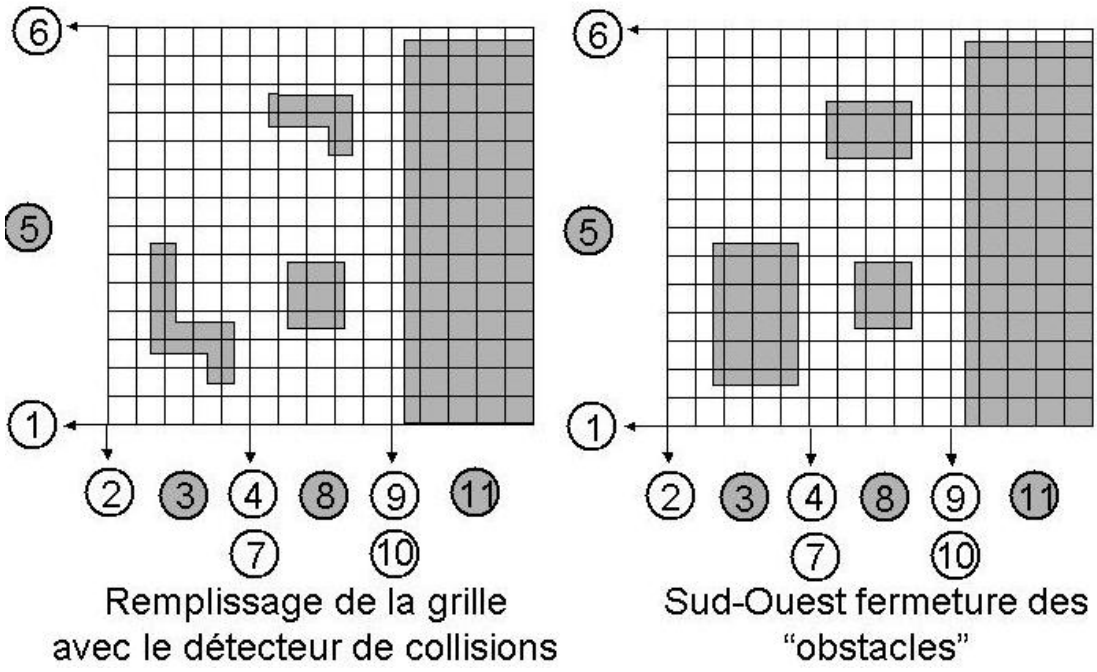

FIG. VI.4 – Remplissage de la grille et fermeture Sud-Ouest des obstacles de collisions.

modèle instantanée : dans ce modèle, les accélérations ne sont pas bornées. Chaque segment du chemin trouvé indique les différentiels de vitesse entre chaque robot. Nous attribuons sur chacune des arêtes du chemin sur la grille des vitesses pour chacun des robot. Nous attribuons sa vitesse maximum au robot qui mettrait le plus de temps à parcourir son chemin avec sa vitesse maximum. Cette attribution des vitesses est optimale en temps pour un chemin donné dans la grille[2]. Une illustration de cette procédure est visible sur la figure VI.5.

Discussion

Notre solution de synchronisation est relativement naïve. En effet, le modèle instantané est un peu faible. Malgré tout, nous rappelons que cette thèse porte sur la planification, et que ce champ d'étude ne vise pas à trouver une solution à un problème en prenant toutes les contraintes en compte.

Toutefois, même en restant sur le modèle instantané, la non prise en compte de la temporalité des actions symboliques est problématique. A l'heure où nous écrivons ces

[2]La démonstration par l'absurde de ceci est aisée.

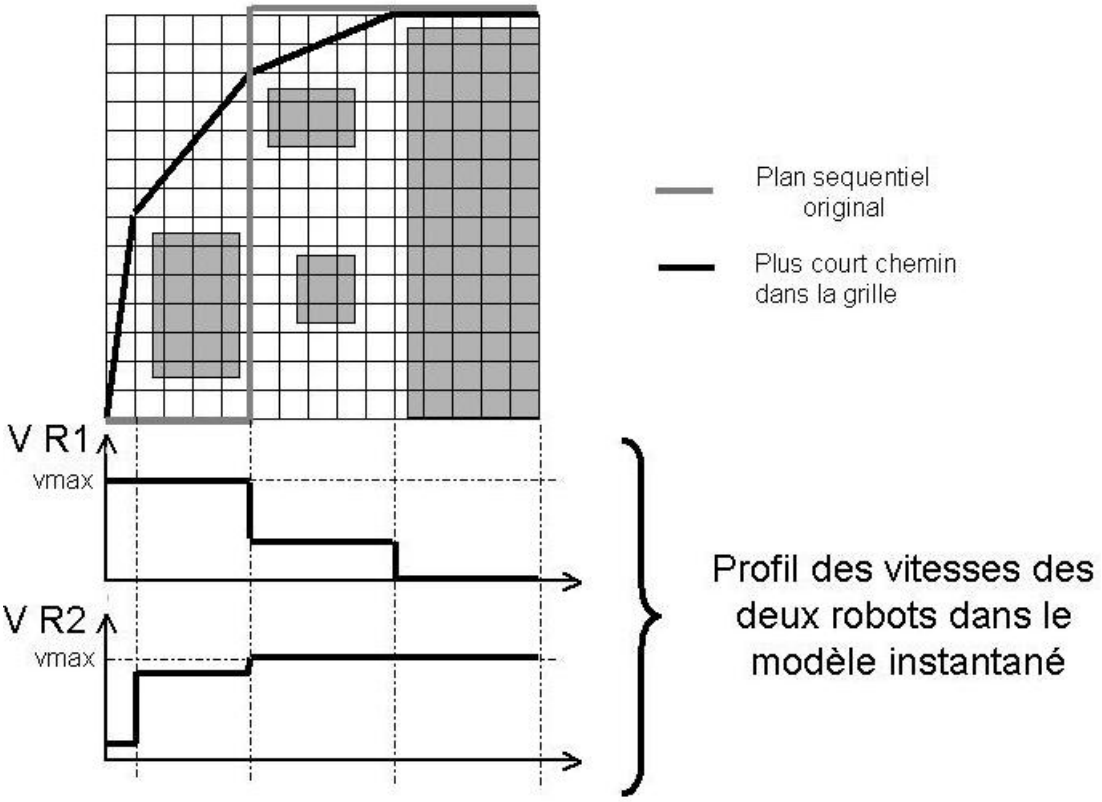

FIG. VI.5 – Solution de synchronisation dans le modèle instantané.

lignes, nous n'avons pas trouvé de solution s'intégrant bien avec l'approche montré plus haut. La non-représentation explicite du temps dans la grille en est le principal obstacle.

Un autre problème, plus formel, se résume avec la question suivante : est-ce que la méthode proposée ci-dessus donne une temps d'exécution du plan optimal dans le modèle instantané pour un pas de discrétisation de trajectoire donnée ? A l'heure où nous écrivons ses lignes, nous n'avons pas réussi à prouver

Un autre aspect concerne la prise en compte plus fine des contraintes dynamiques des systèmes mobiles. Nous pensons que considérer les robots comme des points évoluant sur des courbes avec des bornes en vitesses et en accélération n'est pas suffisant. Il faudrait pouvoir prendre en compte toutes les contraintes dynamiques dans une seconde étape comme par exemple le respect du non-glissement. On se rapprocherait alors de méthode de planification de mouvement prenant en compte les contraintes cinématiques dans un premier temps et les contraintes dynamiques dans un second

temps [Kuffner 98]. Le profil de vitesse dans le modèle instantané permettrait peut-être de guider une recherche de solution respectant les contraintes dynamiques. Une étude plus poussée de cette problématique nous rapprocherait plus d'un problème d'exécution de plan.

Chapitre VII

Résulats et évaluations

Dans ce chapitre, nous tentons d'évaluer le système. aSyMov étant le seul planificateur à notre connaissance capable de résoudre la problématique introduite au début de cette thèse, son évaluation ne peut pas être comparative. Dans la première partie de ce chapitre, nous montrons que le système a quelques bonnes propriétés. Cette partie nous servira à montrer la pertinence de notre implémentation. Dans la deuxième partie, nous donnerons des résultats quantitatifs globaux et nous tenterons de définir qualitativement les limites du système.

VII.1 Bonnes propriétés du planificateur aSyMov

Nous donnons ici une évaluation de nos heuristiques et de nos méthodes algorithmiques. Tout d'abord nous vérifierons que notre méthode de sélection d'action est crédible pour cela nous évaluerons les composants de notre heuristique. Ensuite nous justifierons l'expansion des roadmaps en cours de recherche par deux mesures. Enfin, nous finirons par extraire une mesure démontrant que le planificateur n'explore qu'une petite partie des mouvements possibles.

VII.1.1 Une bonne sélection des actions

Nous avons voulu vérifier que notre approche de sélection d'action était correcte. Nous avons choisi une variante du problème des chariots élévateurs (I.2.1). Dans cette variante, le but est de transporter la petite boîte d'un côté à l'autre de l'environnement. Les prises discrètes sont limitées à une seule prise. Sur ce problème précis, nous avons deux contraintes génantes. Premièrement, la petite boîte ne peut être prise que par le chariot 2 lorsqu'elle est en configuration initiale et ne peut être déposée que par le chariot 1 en configuration finale (voir figure VII.1). Cette contrainte n'est pas représentée dans la partie symbolique mais peut être détectée par une analyse des G.C.E. Notre heuristique générale le prend en compte.

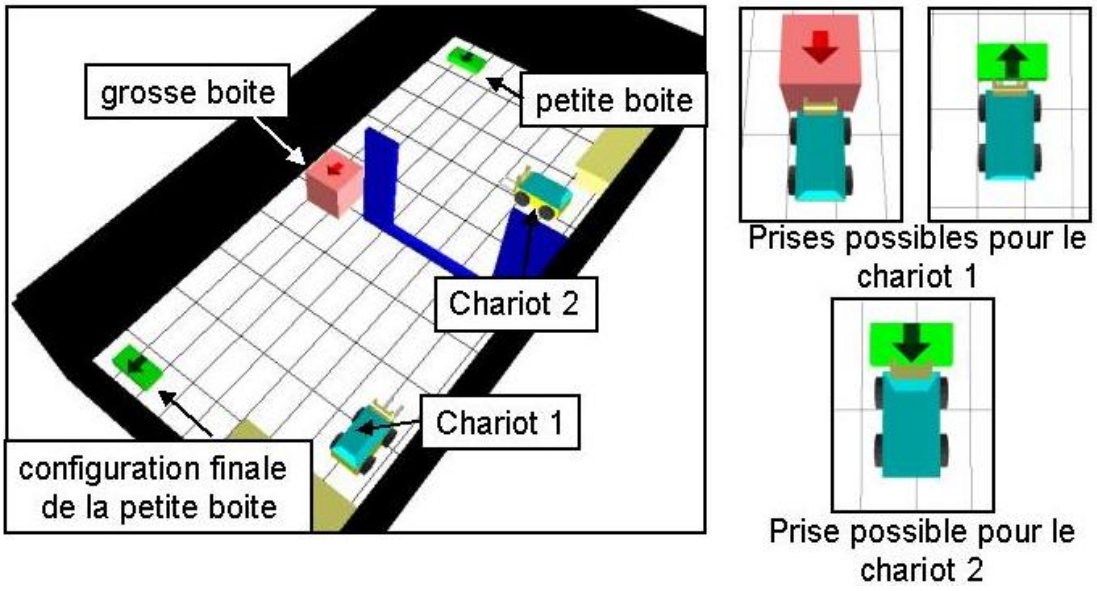

FIG. VII.1 – Une instance particulière des "chariots élévateurs".

Deuxièmement, la grosse boîte doit être déplacée pour laisser le passage au robot. Cette contrainte n'est pas représenté au niveau symbolique, de plus elle est indétectable du point de vue des G.C.E. C'est typiquement le genre de contrainte "piègeant l'heuristique". Nous avons volontairement pris un problème de ce type pour vérifier que nos heuristiques offre tout de même un bon guidage.

Nous faisons résoudre ce problème par aSyMov avec des roadmaps déjà calculés et en débranchant la possibilité d'ajouter de nouveaux noeuds. Vu que la sélection des actions est probabiliste, nous faisons résoudre 50 fois le problème et nous faisons

configuration d'heuristique	(1)	(2)	(3)	(4)	(5)
nombre d'étapes moyen	422	318	288	231	192
temps de calcul moyen	1,38	1,06	0,72	0,54	0,60

TAB. VII.1 – Résultats des recherches pour 5 configurations de l'heuristique de choix de l'action.

la moyenne des paramètres de sortie qui nous intéressent : le temps de calcul[1] et le nombre d'étape de recherche (nombre d'appel à la procédure d'extension d'état). Nous avons testé les cinq configurations commentées ci-après. Les résultats sont résumés dans le tableau VII.1.

– Sans heuristique (1) : à chaque sélection d'action, toutes les actions applicables ont une chance équiprobable d'être sélectionnées. aSyMov profite tout de même du découpage symbolique des actions et de la recherche en avant de solution. Il est de savoir qu'elle est l'influence de ceci sur le résultat.

– Sans heuristique mais avec les coûts des échecs (2) : seul les coûts des échecs sont pris en compte. La résolution se fait alors en 1,06 seconde pour 317,9 étapes.

– Avec heuristique purement symbolique (3) : nous débranchons juste la correction par la géométrie de la longueur du plan symbolique. Le coût des échecs est pris en compte. La résolution dure 0,72 seconde pour 288,46 étapes de calcul. Nous constatons que notre idée d'heuristique basée sur un plan symbolique semble correcte. Surtout que le premier plan symbolique proposé est très éloigné du plan final. Mais nous pensons que la représentation symbolique nous permet tout de même de coder implicitement des procédures qui vont guider la recherche. Ces procédures sont par exemple : pour déplacer un objet il faut s'en approcher, le saisir, le déplacer et le reposer... Un test similaire sur un problème possédant de nombreuses contraintes relationnelles comme celui des tours de Hanoi aurait clairement amené un bien meilleur résultat.

– Avec heuristique générale (4) : les coûts tels qu'ils ont été décrit en V.2.4. Résolution en 0,54 seconde et 231,5 étapes. La correction à l'heuristique basée sur les G.C.E. prouve ici son efficacité.

– Avec heuristique couplée (5) : nous prenons la configuration (3) "avec heuristique purement symbolique" et nous intégrons directement les connaissances topologiques que le planificateur peut connaître avec une analyse des G.C.E dans le problème symbolique. Ce procédé a été mis en place relativement simplement au niveau du calcul des actions applicables du planificateur de tâches. Une action devient applicable que si elle ne correspond qu'à une seule étape de diffusion au sein

[1]Tous les temps de calculs que nous présenterons ont été obtenus sur Pentium 4 HT 3,2 GHz.

roadmap	transit	transfert Small	transfert Medium	transfert Big
nombre de mouvements	7,04	4,02	2,16	1,0
nombre de noeuds	194,9	43.86	23.52	15,68

TAB. VII.2 – Nombre de noeuds développés dans chacune des roadmaps de mouvements pour le problème (1) des tours de Hanoï.

du G.C.E. Cela a pour conséquence d'interdir la production de plans possibles où le chariot 2 dépose la petite boîte dans sa position finale. La résolution se fait en 0,6 secondes mais avec seulement 192 étapes de calculs. Il semblerait donc qu'une approche qui calculerait un plan symbolique en prenant en compte directement les impossibilités issues de la géométrie soit une meilleure approche. Nous discuterons de ceci en conclusion de cette thèse.

Un exemple de plan solution à ce problème peut être visualisé sur notre site internet http ://www.laas.fr/˜scambon.

VII.1.2 Intérêts de l'expansion de roadmaps en cours de recherche

L'expansion de noeud en cours de recherche présente deux intérêts majeurs déjà évoqués en V.2.6. Nous allons montrer par l'expérimentation leurs réalités.

Le **premier intérêt** est que nous pouvons choisir les roadmaps à étendre suivant l'intérêt qu'elles peuvent avoir pour la réussite de la planification. Avec cette méthode on limite ainsi la taille des roadmaps peu ou pas utiles étendre plus largement les roadmaps plus utiles. Pour démontrer ceci par l'expérience, nous avons analysé le nombre de noeuds développés dans les roadmaps du problème (1) des tours de Hanoï . Dans ce problème nous avons quatre roadmaps de mouvements principales : la roadmap de transit caractérisant les mouvements du robot seul et les trois roadmaps de transfert caractérisant le mouvement du robot transportant un des cylindres. Dans le tableau VII.2, nous compilons les résultats obtenus en faisant la moyenne sur 50 planifications.

Dans la table VII.2, le champ "nombre de mouvements" correspond à la moyenne du nombre de mouvements constatés dans les 50 plans obtenus. La solution optimale fait appel à 7 mouvements de transit, 4 mouvements de transfert du cylindre Small, 2 mouvements de transfert du cylindre Medium et 1 mouvement de transfert du cylindre Big. On peut constater que le nombre de noeuds des roadmap est bien en adéquation avec le nombre final de mouvements. On peut toutefois noter que le problème (1) des tours de Hanoï est un problème "idéal" dans le sens où aSyMov est très bien guidé par sa composante symbolique et où l'environnement géométrique est peu contraint.

Autre remarque, le nombre de noeud de transit est plus grand proportionnellement au nombre de mouvement. Cela s'explique par les ajouts de noeud que provoquent les roadmaps relatives. Les résultats restent alors conformes dans le sens où seuls les mouvements de transit font appel à des mouvements d'approches relativement contraints. Ils nécessitent donc un plus grand nombre de noeuds.

Le **second intérêt** à l'expansion des roadmaps en cours de recherche est la possibilité de rajouter des noeuds en prenant en compte un contexte géométrique possible. Comme nous l'avons expliqué en V.2.6, lors d'une application d'action d'expansion de roadmap, nous prendrons un des états géométriques de l'état d'aSyMov courant comme contexte de collision. Lorsque nous débranchons cet aspect dans aSyMov, c'est à dire que l'expansion de roadmap ne prend pas en compte les autres objets, les performances chutent considérablement. Nous avons testé ceci sur le problème (1) des tours de Hanoï. Avec la prise en compte du contexte de collision nous obtenons une performance moyenne sur 50 résolutions d'à peu prés 10 secondes de temps de calculs. Les résultats sont compris entre 1,4 secondes et 38,6 secondes. Lorsque nous débranchons la prise en compte du contexte, force est de constater qu'il est très difficile de trouver une solution, nous avons arrêté la recherche après une heure de calcul pour la première résolution.

La prise en compte du contexte de collision est donc une composante obligatoire pour le succès de la planification. Ces contextes ne peuvent être connus que grâce à l'expansion des roadmaps en cours de recherche.

VII.1.3 Une bonne restriction du nombre de mouvements étudiés

Cette dernière évaluation spécifique va nous donner une estimation du nombre de mouvements étudiés pendant une résolution de problème. Dans le chapitre II nous avons affirmé que la seule hiérarchie possible entre un planificateur de tâches et un planificateur de mouvement est la moins évidente a priori. En effet, une "bonne" hiérarchie (mais inatteignable pour les problèmes réalistes) ferait d'abord appel à un planificateur de mouvements pour étudier la faisabilité de tous les mouvements possibles au niveau géométrique puis transformerait les données obtenues en prédicats spécifiant la connexité entre les divers lieux du problème. Nous appelons ces prédicats, les prédicats `path`.

Nous voulons ici évaluer le nombre de prédicats `path` dont aSyMov a réellement étudié la valeur de vérité. Si d'un point de vue symbolique nous avions voulu représenter toutes les interactions prises en compte par aSyMov, nous n'aurions pas pu nous restreindre à un prédicat `path` mettant seulement en relation la position d'arrivée et la position de départ. Ce prédicat devrait également prendre en considération la position des autres robots et objets. En d'autres termes dire qu'il y a un chemin entre un point

Instance du problème	nombre de prédicat nb *path* étudié	nombre de prédicat nb *path* nécessaire au plan	nombre de prédicat nb *path* total
(1)+roadmaps	17.5	14	151
(1)	25.4	14	151
(4)	71.2	22	151
(2)	74.4	?	16148

TAB. VII.3 – Nombre de prédicats `path` étudiés dans des instances de problème des tours de Hanoï.

A et B ne suffit pas, il faut dire qu'il y a un chemin entre un point A et B pour le robot R1 avec le robot R2 en un point C, l'objet O1 en un point D, etc... Nous voulons donc compter le nombre de tels prédicats réellement étudiés.

Le problème des tours de Hanoï est le plus adapté à cette étude. Pour résoudre ce problème, nous avons fortement réduit le facteur de branchement en interdisant au robot seul de se déplacer vers une pile vide et en interdisant au robot transportant un cylindre de se rendre vers une pile où il ne pourra pas déposer le cylindre. Dans ces conditions, le nombre de prédicats `path` en prenant en compte les positions des disques est de 151[2]. Nous voulons alors savoir quelle proportion de ces prédicats a été réellement étudié. Pour cela nous étudions les actions soumises à la validation (V.2.5) durant une planification d'aSyMov. Les résultats sont donnés sur la table VII.3.

Nous avons étudié cet aspect sur diverses instances du problème (illustré figure I.3) des tours de Hanoï détaillée ici :

– **(1)+roadmap** : l'instance (1) du problème (voir figure I.3) avec de roadmaps calculés avant la planification. Pour ce problème, le nombre de prédicat `path` utilisé dans le plan est de 14.

– **(1)** : l'instance (1) sans roadmap de départ. Le nombre de prédicat `path` utilisé dans les plans solutions est de 14.

– **(2)** : l'instance (1) avec deux robots. Le nombre de prédicat `path` utilisé dans le plan dépend de la solution trouvées. Il est en général inférieur à 14 car le second robot sert en quelque sorte de quatrième pile. Le nombre de prédicats possibles par contre est beaucoup plus important, le second robot apporte beaucoup plus de combinatoire. Le nombre de prédicats possibles a été calculé "à la main".

– **(4)** : la variante (4) sans roadmap de départ. L'obstacle central ajoute des contraintes géométriques qui ne permettent pas de trouver une solution en 14 mouvements. Ces contraintes sont illustrées sur la figure VII.2. Ces contraintes sont inconnues d'aSyMov au début de la planification.

[2] décomptés à la main

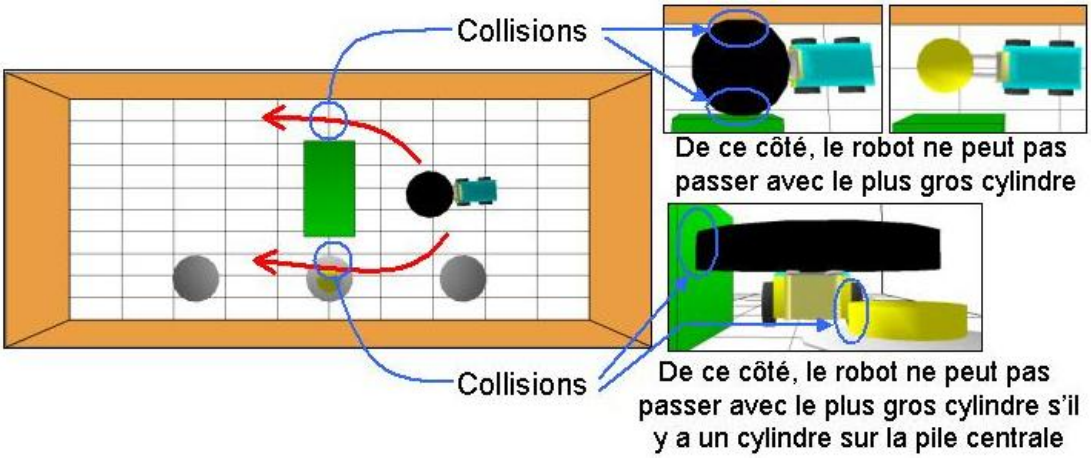

FIG. VII.2 – Les contraintes géométriques de la variante (4) des tours de Hanoi.

Les résultats sont toujours une moyenne sur 50 résolutions. Sur cette table, nous considérons que nos résultats sont satisfaisants. Un des buts d'aSyMov était bien de limiter la combinatoire qu'implique forcément la prise en compte des contraintes géométriques d'un problème. On voit bien par contre sur l'instance (4) que lorsque le plan symbolique solution n'est pas en accord avec une solution possible, le planificateur va essayer d'autres "voies". Dans le cas des deux robots, le nombre de prédicats `path` que le planificateur ne va pas étudier est encore plus important. Nous avons une bonne restriction du nombre de mouvements étudiés.

VII.2 Résultats quantitatifs, limites du système

Dans cette partie, nous allons tenter d'évaluer les limites du système. Quels sont les problèmes que aSyMov peut résoudre dans un temps raisonnable.

VII.2.1 Études des limites du système

Pour étudier les limites du système nous allons comparer plusieurs variantes du problème des chariots élévateurs (cf. I.2.1). Pour cela nous allons faire varier le nombre de robots et le nombre de boîtes à déplacer. Nous nous intéressons aux problèmes suivants :

– *"1 chariot 1 boite"*. Un chariot doit transporter une boîtes de sa position initiale à sa position finale. Le problème qui nous a servi d'exemple pour la validation (cf. V.2.5) fait parti de ce type de problème.

– *"1 chariot 2 boites"*. Un chariot doit transporter deux boites dans leurs positions finales.

– *"2 chariots 2 boites"*. Deux chariots doivent transporter deux boites de leurs positions initiales à leurs positions finales.

Pour l'expérimentation nous faisons 20 tests pour chaque type de problème. Pour chaque résolution, nous tirons aléatoirement les configurations initiales et finales des boites et des robots. Si la configuration finale ou la configuration finale est en collision, nous écartons ce problème : il n'a pas de solution. Nous considérons sans vérification que tous les autres problèmes sont résolubles.

Souvent dans ces problèmes, les boites doivent être déposées en position intermédiaire pour changer de prises. En effet les orientations initiales et finales des boites ne permettent pas toujours un transfert direct. Pour aider le planificateur, nous lui donnons cette information en contraignant le problème symbolique pour que tous les plans possibles qui seront proposé fasse passer les boites par des positions intermédiaires. Si cela n'était pas utile, la validation "gommera" les actions supplémentaires en ne les rattachant pas à des chemins superflus. Pour s'en convaincre, le lecteur peut reprendre la résolution proposée en V.2.5 en supposant que l'état final de la boite est atteignable en un seul transfert depuis l'état initial.

De manière générale, les informations sur le domaine qu'un expert humain peut détecter seront transmises à aSyMov sous forme de contraintes symboliques. Nous faisons cela sans états d'âme : la composante symbolique d'aSyMov sert justement à limiter la recherche sur la base d'informations que les utilisateurs du système peuvent identifier.

Autre remarque, aSyMov est composé de beaucoup de paramètre (nombre de cycle dans les algorithmes, valeur heuristiques par défauts, influence de la taille des roadmaps sur les coûts des règles heuristiques, etc). Il est donc difficile de faire une étude complète de l'influence de tous les paramètres sur les performances du planificateur, surtout que de bons paramètres pour un problème donné peuvent se révéler désastreux pour un autre problème. Par exemple pour le problème "1 chariot 1 boîte", le temps moyen de résolution en utilisant l'algorithme "Visibilité - PRM" pour l'extension de la roadmaps de transit et de la roadmap de transfert met en moyenne 10 fois plus de temps de calcul qu'avec une utilisation de l'algorithme PRM classique. Dans ce problèmes, vu que l'expansion de la roadmap de transit ne prend pas en compte le placement de la boîte, la visibilité est délicate à utiliser. Ce n'est pas le cas par exemple pour les problèmes des tours de Hanoï, car les placements des objets sont très limités.

Les résultats donnés font donc intervenir 'algorithme PRM classique. De la même manière, nous gardons les mêmes paramètres pour les trois types de problèmes.

Les résultats sont compilés dans la table VII.4. La recherche est arrêtée après 500 secondes de calcul. Les résultats de temps de calcul ne sont donnés que pour les

Problème	temps de calcul (s)	taux de réussite
1 chariot 1 boite	12,87	100%
1 chariot 2 boites	49,94	45%
2 chariots 2 boites	148,82	15%

TAB. VII.4 – temps de calcul moyen pour 3 types de problèmes “chariots élévateurs”.

problèmes dont le planificateur a trouvé une solution.

Plus nous considérons de robots et d’objet différents, plus une solution semble difficile à trouver. Cela est dû principalement au nombre d’interaction qui augmente exponentiellement avec le nombre de robot et d’objet ce qui se traduit par une augmentation exponentielle du nombre de plans possibles. Par exemple dans le problème “1 chariot 1 boîte”, tous les plans possibles commence par la même action : celle d’aller dans une position permettant de saisir la boîte. Dans un problème “2 chariots 2 boîtes”, quatre actions peuvent commencer un plan possibles. Il semblerait que le facteur de branchement du problème soit un des paramètres les plus néfastes.

Une illustration de ceci est donné sur les figures VII.3 et VII.4[3]. Ces deux figures, issues de notre outil de visualisation de l’avancement de la recherche, représentent l’espace de recherche exploré pour deux problèmes différents. Sur ces graphes nous représentons tous les états ayant le même état symbolique par le même noeuds. Les plans possibles peuvent être extraits en suivant un chemin dans ces graphes allant de l’état initial à un état final en suivant les arcs orientés. Les noeuds gris foncé sont les états initiaux et finaux. Les noeuds gris sont ceux dont le planificateur a trouvé au moins un état géométrique correspondant à l’état symbolique. Les flèches fines sont des actions symboliques que le planificateur n’a pas tenté de valider. Les flèches grasses représentent des actions qu’aSyMov a tenté de valider. Ces figures illustrent bien l’explosion de l’espace de recherche lorsque le nombre de robots ou d’objets augmente.

Lorsque le planificateur est en train de se “perdre”, il a tendance à ajouter de plus en plus de noeuds dans les roadmaps. Cela augment le temps de calcul passé dans la validation des actions. Nous avons tenté de pallier à ce problème en ajoutant une action de “filtrage” des roadmaps. Cette action de filtrage des roadmaps, au même niveau que les actions applicables et les actions d’expansion de roadmaps, éliminent les noeuds et les arcs semblant “inutiles”, comme par exemple si ils n’ont pas encore été utilisé pour valider une action. Nous n’avons pas obtenus de résultats satisfaisant avec cette tentative d’amélioration.

[3]Nous ne demandons pas au lecteur d’essayer de lire les notations sur ces graphes...

	taux de réussite	temps CPU(s)	nb action	nb action optimal
(1)+roadmaps	100 %	0,266	28	28
(1)	100 %	9,97	28.8	28
(2)	78 %	78,2	16,3	?
(3)	100 %	298,9	40,16	28
(4)	100 %	271,6	44	44

TAB. VII.5 – Résultats moyens sur plusieurs instances du problème des tours de Hanoï.

VII.2.2 Résultats sur notre jeu de problèmes

Pour le problème interrupteur/radio, le temps de calcul moyen pour 20 résolutions est de 18.6 secondes. D'un point de vue géométrique, ce problème n'est pas foncièrement différent d'un problème de type "chariots élévateurs" avec un robot et une boîte.

Pour les problèmes des tours de Hanoï, nous avons compilé des résultats moyens sur 50 tests dans la table VII.5. La recherche est arrêtée si aucune solution n'a été trouvée après 3 heures de calculs. Les moyennes de temps de calcul sont donnés pour les planifications réussies.

Ce problème se prête bien au planificateur aSyMov. Nous réussissons en général à trouver une solution. Ce problème est en fait relativement simple car les objets ne peuvent pas être placés n'importe où dans l'environnement. Cela limite grandement l'espace des configurations associé au problème.

Pour la variante (3), l'obstacle central est volumineux mais ne provoque pas les mêmes contraintes géométrique que pour la variante (4) (cf. figure VII.2). Dans cette variante, aSyMov peut trouver des plans similaires à ceux de la variante (1) ou des plans similaires à ceux de la variante (4). C'était une surprise pour nous.

Quant au problème IKEA, force est de constater que ce problème est trop complexe pour être résolu entièrement automatiquement par aSyMov. En effet, nous avons deux robots et trois objets. De plus les objets peuvent se composer entre eux, par exemple le pied assemblé à la planche forment un nouvel objet. Pour ce problème nous avons donc mis en place plus de 40 roadmaps. Certains mouvements nécessaires sont très contraints, en particulier celui du retournement de la table par les deux robots. Ce sous problème en lui-même est difficile. Nous avons donc construit certaines de ces roadmaps "à la main".

Malheureusement, même avec des roadmaps précalculées, il reste très difficile de trouver une solution dans un temps raisonnable. La présence de 5 robots/objets et la longueur du plan solution sont les principaux obstacles à ceci. Nous avons tout de même réussi à trouver des solutions à ce problème en aidant le planificateur à faire les

bons choix au moment cruciaux de la recherche. Une d'entre elle peut être visualisée, comme les autres solutions sur le site web http ://www.laas.fr/~scambon.

Le problème IKEA est surtout pour nous une manière de monter la richesse de la représentation développée au cours de cette thèse. La résolution en initiative mixte prouve que notre planificateur possède tous les outils pour "digérer" ce genre de problème. Sa résolution de manière totalement automatique est un défi pour le future.

VII.2.3 Conclusion sur l'évaluation

aSyMov présente de bonnes propriétés présentées en début de chapitre. Les idées qui nous ont guidées dans sa conception sont empiriquement validées. Toutefois l'espace de recherche de problème même simple en apparence est très grand. C'est surtout le cas quand le problème met en jeu plusieurs robots pouvant accomplir les mêmes tâches car cela fait augmenter exponentiellement le nombre de plans possibles. La dimension intrinsèque de l'espace de configurations considéré se retrouve alors dans la combinatoire des plans possibles. Les autres limites sont les mêmes que pour un planificateur de manipulation : la présence d'espaces contraints, la continuité des placements des objets et les prises continues handicapent la recherche d'une solution. aSyMov hérite également des limites du planificateur de tâches utilisé.

Le système obtient de bons résultats lorsque les problèmes mettent en jeu des robots hétérogènes n'ayant pas le mêmes aptitudes (comme dans le problème interrupteur/radio). Il obtient également de meilleures résultats quand :

- Le problème symbolique et le problème global ont des solutions proches.
- Les placements des objets sont limités.
- On utilise des prises discrètes.
- L'environnement géométrique est peu contraint.

Par rapport à un planificateur de tâches, les plans que nous produisons sont plus riches. Ils prennent explicitement en compte les contraintes liées aux mouvements et aux manipulations. Ces plans auront une meilleure aptitude à guider les robots dans leurs décisions. De plus les plans ne produisent pas seulement un ordre d'actions mais aussi une manière de mener à bien les actions de mouvements et de manipulation. Les plans produits pour les problèmes des tours de Hanoï illustre bien ceci, nous pensons qu'une réelle étape a été franchi ici par rapport aux plans produits par un planificateur de tâches classique.

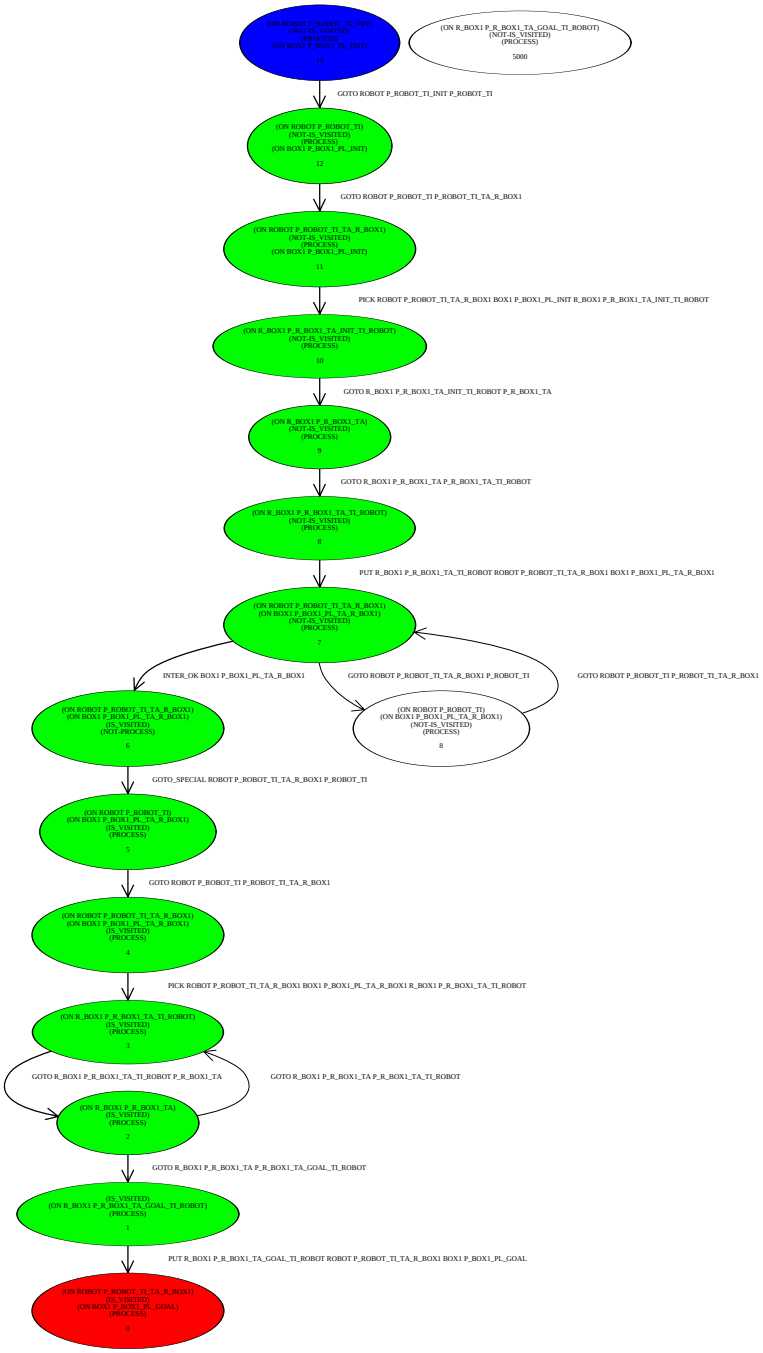

FIG. VII.3 – L'espace de recherche exploré pour un problème "1 chariot 1 boîte".

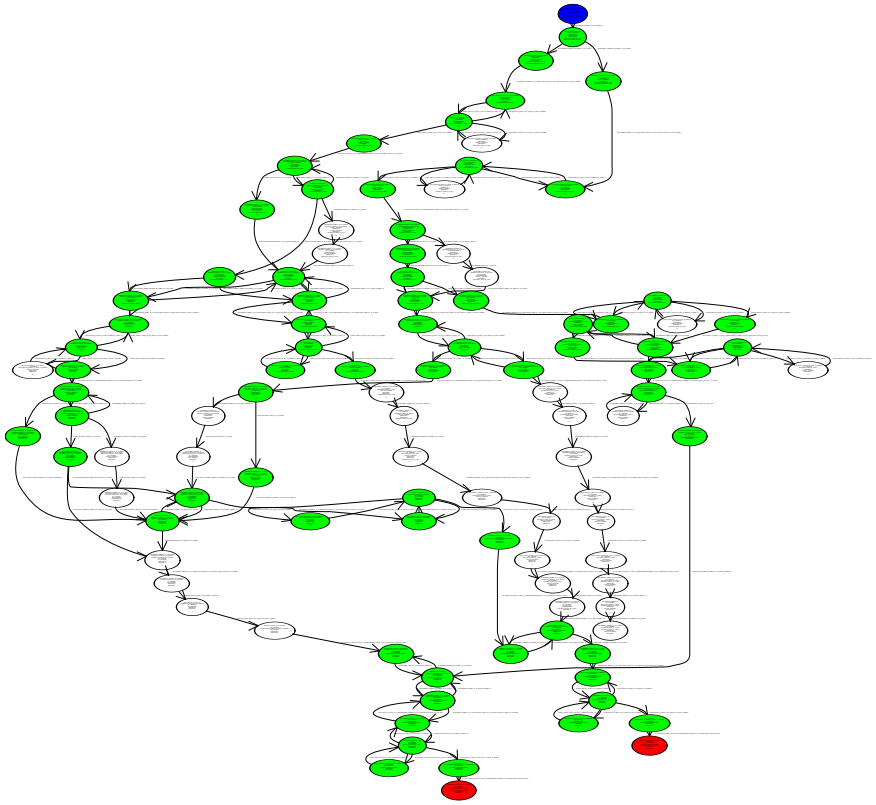

FIG. VII.4 – L'espace de recherche exploré pour un problème "1 chariots 2 boîtes".

Chapitre VIII

Conclusion : aSyMov et le monde réel

Ce chapitre est plus prospectif et fera office de conclusion. Nous revenons ici sur les besoins de prise de décision pour les robots. Nous ouvrirons ensuite sur nos travaux en cours pour améliorer notre système à ces besoins.

VIII.1 Les besoins réels en robotique

Nous voulons maintenant revenir sur la prise de décision pour les robots autonomes. Pour appuyer notre réflexion, nous proposons ci-après un modèle conçu pour une application de robotique réelle.

VIII.1.1 Un exemple “monde réel”

Sur la figure VIII.1.2 nous illustrons un modèle auquel nous nous sommes intéressés pour montrer que nous pouvons nous attaquer à des problèmes pertinents pour les expériences que le roboticiens veulent mettre en place.

La plupart des données du modèle sont des données “réelles”. Le robot existe réellement au sein de notre laboratoire et répond au nom de “Hilare”. Les obstacles statiques proviennent directement d’une carte construite par le robot grâce à ses télémètres

lasers. Hilare possède deux télémètres lasers, un à l'avant du robot et un à l'arrière situé sur sa remorque. La carte est celle du LAAS-CNRS, notre laboratoire. Sur la figure, nous sommes plus exactement situé dans la "grande salle" robotique.

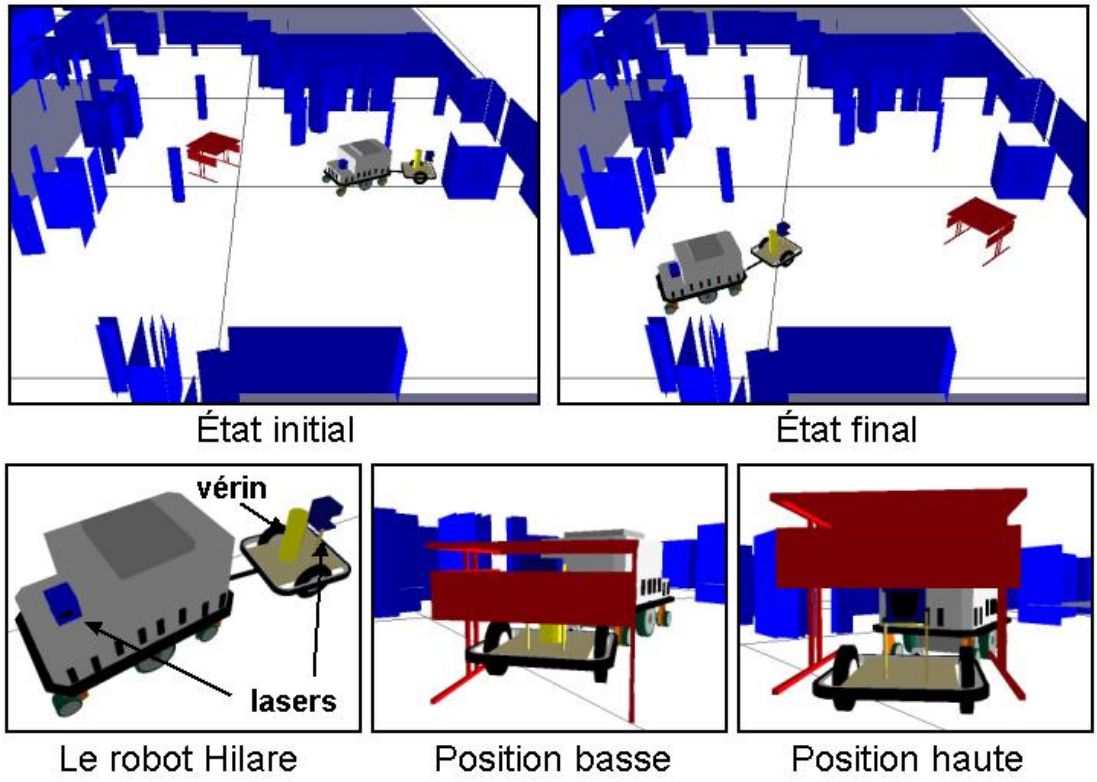

FIG. VIII.1 – Le modèle et les contraintes pour hilare transportant une table.

Ce qui n'existe pas dans la réalité, c'est l'ajout d'un vérin sur la remorque du robot. Nous n'avons d'ailleurs pas fait d'effort particulier de conception pour celui-ci : il se résume à un cylindre pouvant être levé ou abaissé. Nous souhaitons qu'Hilare puisse accomplir une tâche simple de manipulation d'une table grâce à son vérin. La table à été conçue pour intercepter les segments lasers lorsque celle-ci est posée au sol, le robot pourra alors se positionner par rapport à celle-ci. Par contre une fois qu'Hilare est en position pour transférer la table, il doit la soulever pour ne pas obstruer son télémètres laser arrière. Ce mécanisme est très important pour la localisation du robot dans son environnement.

Les avantages d'aSyMov sont clairs pour ce genre d'application. En effet on peut

coder la contraintes de levage de table et de maintien de la table en position haute lors des mouvements de transfert dans la partie symbolique de la représentation. Ainsi, quelles que soient les configurations initiales et finales de la table et du robot, aSyMov pourra produire un plan calculant les mouvements en respectant les contraintes évoquées. De plus, aSyMov fera respecter les contraintes géométriques tridimensionnelle que peuvent contenir le modèle (pour peu que la "carte" ait été réalisée en trois dimensions...). Si par exemple, il y a dans l'environnement un passage trop bas pour que le robot puisse passer en position "table levée", le planificateur proposera un plan contournant ce passage. Le modèle étant conçu, il n'y aurait pas non plus de difficulté à changer les dimensions de la table, du robot ou de l'environnement statique : aSyMov pourra toujours tenter de trouver des plan solutions. Pour la conception de ce modèle, nous profitons de travaux sur l'étude des mouvement des véhicule à remorque ([F.Lamiraux 99]).

VIII.1.2 Discussion

Comme nous l'avons souligné au début de cette thèse, un plan n'est qu'un guide permettant la décision de robots autonomes. Les incertitudes existant dans le monde "réel" sont nombreuses. Cela est dû notamment aux imperfections de la localisation du robot dans son environnement, les lois de commande du robot ne vont pas forcément l'amener précisément à la position souhaitée, etc. La carte utilisée par le robot peut se retrouver obsolète : un meuble a été déplacé, un humain a laissé malencontreusement traîner un objet sur le sol, etc.

Pour toutes ces raisons, le robot doit être capable de modifier sa carte et de vérifier que le plan initialement proposé est toujours valide malgré la récolte de nouvelles informations. Tout ce travail n'est pas forcément dévolu au seul planificateur, on peut tout d'abord tenter de modifier localement le plan avant de le remettre complètement en cause. On parle alors de procédures réactives. Par exemple, le robot Hilare est équipé d'algorithmes de déformation de trajectoire lors d'une détection d'obstacle non prévu dans l'environnement ([F.Lamiraux 02]. Dans le cas où la déformation de trajectoire échoue, il faudra faire appel au planificateur.

Plusieurs appels successifs au planificateur peuvent donc être effectués pour des calculs de plan relativement similaires. Nous avons testé notre implémentation actuelle sur la résolution du problème présenté sur la figure . Sur 20 résolutions, la planification réussie dans 50% des tests en dessous des 500 secondes pour un temps CPU moyen de 98,5 secondes. Pour ce type d'application cela n'est pas satisfaisant. Le moindre appel au planificateur en cours d'exécution peut prendre de longues minutes de calcul.

VIII.1.3 Améliorations envisagées

aSyMov a été initialement conçu pour résoudre un problème sans connaissances au préalable de la topologie. Dans le cas de replanification, cela paraît sous optimal. Nous pourrions nous "resservir" des roadmaps déjà calculé.

L'idéal serait pourtant l'utilisation de roadmaps garantissant une caractérisation "suffisante" de la topologie. Si nous y parvenions, les améliorations apportées à aSyMov seraient nombreuses. Nous pourrions par exemple revenir à une gestion beaucoup plus conventionnel du front de recherche. Malgré tout, nous ne croyons pas que l'obtention de d'une structure garantissant une caractérisation de l'espace des configurations soit faisable en l'état actuel des connaissances. Nous ne pensons pas pourvoir enlever complètement à aSyMov la possibilité d'explorer plus profondément l'espace des configurations.

Toutefois, l'utilisation de roadmaps pré-calculées peut apporter un gain significatif de performance. Par exemple sur les problèmes "1 chariot 1 boîte", la réutilisation des roadmaps calculés pour la résolution d'un problème quelconque pour la résolution de 20 autres problèmes quelconques multiplie la performance par 7 (1,85 secondes au lieu de 12,87 secondes).

Ce type d'approche devient par contre problématique dans le cas où l'espace de travail a changé entre deux résolutions. Certains arcs de roadmaps peuvent alors devenir non valides. Nous devrions alors faire appel à des techniques de mise à jour des roadmaps [Leven 00] [der Berg 04].

Nous avons plus d'espoir avec une extension du concept de roadmaps relatives. Les roadmaps relatives peuvent se rapprocher du concept de recherche locale alors que les roadmaps de mouvements classiques de celui de la recherche globale. La recherche de solution gagne beaucoup à s'appuyer sur ces deux types de recherche. Jusque là, notre utilisation des roadmaps relatives étaient dévolue à l'approche des objets. Nous pensons créer en prétraitements des roadmaps relatives pour des changement de prises par exemple. La recherche d'une configuration intermédiaires pour un objet pourra s'appuyer sur un critère d'adéquation de la configuration de l'objet avec la roadmap de changement de prise. En d'autres termes, l'ajout d'une configuration dans sa roadmap de placement ne pourra se faire que si celle-ci permet un changement de prise via la roadmap relative de changement de prise.

Nous envisageons un autre type d'amélioration, cette fois sur le lien entre la partie topologique et la partie symbolique de la représentation.

Nous avons remarqué que les problèmes de planification de tâches associés à nos problèmes sont résolus rapidement par les planificateurs de tâches modernes. Le temps de résolution est presque négligeable par rapport au temps de calcul passé dans le détecteur de collisions par exemple.

Nous pourrions donc augmenter le nombre de symboles à considérer. Dans notre représentation, plusieurs états symboliques représentent des états d'aSyMov différents. Cela rend la recherche un peu plus délicate dans la mesure ou nous devons autoriser des boucles dans l'espace d'état, ce qui rend l'espace des plans possibles infini.

Ceci est dû à deux paradigmes de représentation différents. D'une part, cela provient de l'ordre des actions précédent la création de l'état d'aSyMov dans le cas de problème mettant en jeu plusieurs robots comme ceci est illustré sur une figure précédente (figure V.3).

D'autre part nous représentons toutes les composantes connexes d'une roadmap par un même symbole. Cela nous paraît maintenant une idée à approfondir surtout en ce qui concerne les différentes prises qu'un robot peut appliquer sur un objet. Nous pensons qu'un découpage symbolique plus en accord avec la topologie des roadmaps aiderait la recherche. Notamment, nous pourrions regrouper nos deux "expertises heuristiques" : celle basée sur la longueur du plan symbolique et celle basée sur l'analyse du G.C.E.

L'idée serait de créer un symbole de position par composante connexe dans les roadmaps. Ainsi, chaque prises seraient différenciées au niveau symbolique. Cette idée semble bonne mais elle n'est pas triviale dans son implémentation.

Premièrement, au départ de la recherche, les roadmaps sont vides et donc nous n'avons aucun symboles donc aucun plan possibles. Comment alors guider la recherche avant d'obtenir un plan possible ? Certainement qu'il faudrait maintenir deux types de représentation symbolique : celle étudié le long de cette thèse pour le début de la recherche et celle proposée ici.

Deuxièmement, nous serions obligés d'ajouter ou de regrouper des symboles en cours de recherche ce qui est la contrepartie des découvertes de nouvelles composantes connexes ou du regroupement de deux composantes connexes au sein des roadmaps. Le point délicat avec ces modifications des symboles à la disposition du planificateur de tâches, c'est que celui-ci serait obliger de résoudre un nouveau problème à chaque changement dans la bibliothèque de symbole. En effet dans l'implémentation actuelle, nos appels multiple au planificateur de tâches bénéficiait du même prétraitement sur l'analyse du domaine de planification. Cela ferait un intéressant sujet de recherche.

VIII.2 Conclusion générale

Au cours de cette thèse, nous sommes partis d'un constat : la faisabilité des mouvements et les prises de décisions de plus haut niveau sont intimement liés pour les systèmes autonomes mobiles. C'est encore plus vrai si ceux-ci sont dotés de capacités de préhension qui leurs permettent de changer la géométrie de l'environnement dans lequel ils évoluent.

Ce constat nous a guidé dans la construction d'un modèle prenant en compte explicitement la géométrie de l'environnement et les conséquences géométriques des actions des robots sur cette géométrie.

Dans cette thèse, nous avons montré formellement comment lier le monde géométrique continu, la cinématique des robots et des représentations relationnelles pour former un problème de planification généralisant la planification de mouvement, la planification de manipulation et la planification de tâches.

La clé ici est d'associer des symboles avec des ensembles continus de configurations possédant une même propriété. Les actions deviennent alors sémantiquement plus riche. Nous considérons que cette idée est le coeur de cette thèse.

Sur cette idée, nous avons construit un planificateur explorant l'espace de recherche ainsi obtenu. Dans notre implémentation, le raisonnement symbolique est basé sur les fonctions d'un planificateur de tâches classique alors que la recherche des configurations appartenant aux ensembles est effectuée par les fonctions d'un planificateur de mouvements probabiliste.

Nous pensons avoir ouvert une nouvelle voie de recherche en regroupant les domaines de la planification de tâches et celui de la planification de mouvements et de manipulations. Notre planificateur n'est qu'un permier prototype capable de raisonner sur des problème imbriquant des tâches et des mouvements. Empiriquement, nous avons réalisé que les problèmes, bien que simples en apparence, sont en fait très complexes. Nous retrouvons a priori la complexité intrinsèque d'un problème de planification de mouvements de haute dimension. En l'état, il semblerait qu'un tel système ne puisse être utilisé que pour des "petites" missions ou pour la résolution de sous problèmes qu'un robot pourrait rencontrer au cours d'une mission. Toutefois, au vue des améliorations qui pourraient être amenées nous sommes convaincus qu'il est possible d'obtenir des réponses rapides du planificateur pour une classe de missions bien maîtrisée.

Nous pensons que de nombreuses applications peuvent s'appuyer sur ce travail en dehors de la robotique autonome mobile comme par exemple pour les personnages animés et l'animation graphique ou l'interaction homme/robot. Pour les personnages animés, nous pourrions nous appuyer sur une représentation géométrique simplifiée, ce qui limiterait la dimension de l'espace de recherche. Dans l'interaction homme/robot, on peut distinguer deux types de contraintes. Il y a des contraintes protocolaires telles que les règles de politesse qui peuvent souvent s'exprimer de manière symboliques. Il y a également des contraintes géométriques lié aux capacités et au confort de l'humain. Par exemple lorsqu'un robot doit amener un objet à un humain, il doit limiter les déplacements de celui-ci, lui tendre de manière explicite. Ces deux types de contraintes ne sont pas indépendante. Par exemple pour dire bonjour à un humain (protocole), il est préférable d'être en face de lui (géométrie). Ces contraintes mixtes peuvent être

gérées par notre planificateur.

Glossaire

Actif : ce dit d'un robot ou d'un objet mis en jeu dans un prédicat `on` dans la partie "état symbolique" d'un état d'aSyMov.

Action : c'est l'instance d'un opérateur. Dans aSyMov, une action a la même signification que pour la planification de tâches classique.

Applicable : se dit d'une action pouvant être appliqué sur un état. Dans aSyMov, ce terme permet de distinguer les actions symboliquement applicable des actions à expansion de roadmaps.

aSyMov : **a Sy**mbolic **Mov**e3d. Planificateur intégrant de manière non hiérarchique un planificateur de tâche symbolique et un planificateur de mouvements et de manipulations.

Composante connexe : c'est l'espace atteignable à partir d'une configuration. Au niveau des roadmaps, c'est un ensemble de noeuds dont on peut toujours trouver un chemin pour relier deux noeuds de l'ensemble.

Connexité : définit l'appartenance à une même composante connexe. Dans une roadmap classique, cette notion prouve l'existence d'un chemin entre deux configurations.

Connexité faible : Notion nécéssaire mais pas suffisante à l'existence d'un chemin entre deux configurations.

Configuration : une instance des degrés de liberté d'un système.

Chemin : courbe dans l'espace des configurations correspondant à un mouvement dans l'espace de travail.

Chemin local : chemin élémentaire entre deux configurations.

Chemin de transfert : chemin d'un robot solidaire avec un objet. Dans un chemin de transfert, la prise du robot sur l'objet reste fixée.

Chemin de transit : chemin d'un robot seul alors que les autres robots et les autres objets sont statiques.

Domaine de planification : Recencement des opérateurs utilisable pour résoudre un problème de planification.

Espace de travail : l'espace réel dans lequel évolue un système mobile. Dans les problème réalistes, il est tridimensionnel.

Espace des configurations : noté *CS*, c'est l'espace incluant toutes les configurations possible pour un système mobile. Cet espace est de dimension égale au nombre de degré de liberté du système considéré.

Espace libre : noté CS_{free}, c'est le sous-espace de l'espace des configurations correspondant à des configurations sans collisions avec l'environnement statique et sans auto-collisions du système avec lui-même.

Liens d'héritage : lien unissant deux roadmaps. Lorsque deux roadmaps ont un lien d'héritage, l'ajout d'une configuration dans une roadmap peut provoquer l'ajout d'une partie de la même configuration dans l'autre roadmap. C'est la projection d'une configuration sur un espace de configuration plus petit.

Opérateur : Action non instanciée.

Position : une position est un type de symbole. D'un point de vue sémantique, un symbole de ce type représente un ensemble de configurations. Dans aSyMov ces symboles sont reliés à des listes de noeuds de roadmap.

Propriété d'une position : une position représente un ensemble de noeuds possédant une même propriété. Nous définissons trois types de propriété : propriété générale ou de généralité, propriété de lien, propriété de zone.

Prédicats spécifiques : ensemble de prédicats nous permettant de représenter de manière relaxé la géométrie du problème et les roadmaps utilisés. Ils sont au nombre de 6 : `belongs-to`, `has-property`, `link`, `compose`, et `on`.

Prise : définit comment un robot peut saisir un objet. Une prise est définie comme une configuration relative de l'objet par rapport au robot. On parle de prise discrète quand on a défini un nombre limité de configuration relative. A contrario pour les prises continues on défini une configuration relative continue entre l'objet et le robot. Le lien entre le robot et l'objet peut alors être vu comme un ensemble de degré de liberté.

Problème de planification : Un état initial et une spécification des états finaux.

Roadmap : graphe dont les noeuds sont des configurations et les arcs des chemins locaux.

Roadmap Probabiliste : roadmap générée par tirage aléatoire des noeuds (i.e. configurations).

Roadmap heuristique : roadmap qui ne caractérise pas des mouvements valides mais qui aide à leurs recherches.

Règles d'enrichissement : règles gérant l'ajout de noeuds dans les roadmaps en cours de planification dans aSyMov.

Types spécifiques : typage de termes minimal devant être utilisé par un développeur de domaine pour aSyMov. Ils y a types spécifiques : `robot`, `obj`, `position`, `subspace` et `property`.

Références bibliographiques

[Akella 02] S. Akella & S. Hutchinson. *Coordinating the Motions of Multiple Robots with Specified Trajectories*. In IEEE International Conference on Robotics and Automation, 2002.

[Alami 89] R. Alami, T.Siméon & J.-P. Laumond. *A Geometrical Approach to Planning Manipulation Tasks. The case of discrete placements and grasp.* In Robotics Research : The Fifth International Symposium, 1989.

[Alami 95] R. Alami, J.-P. Laumond & T.Siméon. *Two Manipulation Planning Algorithms*. In Algorithmic Foundations of Robotics (WAFR'94), 1995.

[Alami 98] R. Alami, R. Chatila, S. Fleury, M. Ghallab & F. Ingrand. *An Architecture for Autonomy*. International Journal of Robotics Research, vol. 17, no. 4, pages 315–337, 1998.

[Backstrom 98] C. Backstrom. *Computational Aspect of Reordering Plans*. Journal of Artificial Intelligence Research, vol. 9, no. 9, pages 99–137, 1998.

[Barraquand 91] J. Barraquand & J.-C. Latombe. *Robot Motion Planning : A Distributed Representation Approach*. International Journal of Robotics Research, vol. 10, no. 6, pages 628–649, 1991.

[Blum 97] A. Blum & M. Furst. *Fast Planning Through Graph Analysis*. Artificial Intelligence, vol. 90, pages 281–300, 1997.

[Brock 97] O. Brock & O. Khatib. *Elastic Strips : Real-Time Path Modification for Mobile Manipulation*. In The English International Symposium of Robotics Research, 1997.

[Brock 02] O. Brock, O. Khatib & S. Viji. *Task-Consistent Obstacle Avoidance and Motion Behaviour for Mobile Manipulation*. In IEEE International Conference on Robotics and Automation, 2002.

[Cambon 02] S. Cambon, S. Lemai & R. Trinquart. *Compétitions de planification*. revue d'Intelligence Artificielle, 2002.

[Cambon 04] S. Cambon, F. Gravot & R. Alami. *A Robot Task Planner that Merges Symbolic and geometric reasonning*. In ECAI'04, 2004.

[Canny 88] J.F. Canny. *The complexity of robot motion planning*. MIT Press, 1988.

[Cortés 03] Juan Cortés. *Motion Planning Algorithms for general closed-chain mechanisms*. PhD thesis, Institut National Polytechnique de Toulouse, 2003.

[der Berg 04] J. P. Van der Berg, D. Nieuwenhuisen, L. Jaillet & M. Overmars. *Creating Robust Roadmap for Motion Planning in Changing Environments*. In IROS 2004, 2004.

[Do 03] Minh B. Do & Subbarao Kambhampati. *Improving the Temporal Flexibility of Position Constrained Metric Temporal Plans*. In Proc. IEEE International Conference on Automated Planning and Sceduling, 2003.

[E.A.Sisbot 04] E.A.Sisbot. *Coopération et Coordination pour un ensemble de robot hétérogène*. Rapport technique, LAAS-CNRS, 2004.

[Fikes 71] R.E. Fikes & N.L. Nilsson. *STRIPS : A new approach to the application of theorem proving and theorem solving*. Artificial Intelligence, no. 2, pages 189–208, 1971.

[Fikes 72] R. Fikes, P. Hart & N. Nilsson. *Learning and Executing Generalized Robot Plans*. Artificial Intelligence, 1972.

[F.Lamiraux 99] F.Lamiraux, S.Sekhavat & J.-P. Laumond. *Motion Planning and Control for Hilare pulling a Trailer*. IEEE Transaction on Robotics and Automation, 1999.

[F.Lamiraux 02] F.Lamiraux & D.Bonnafous. *Reactive Trajectory Deformation for Nonholonomic Systems : Application to Mobile Robots*. In Proc. IEEE International Conf on Robotics and Automation, 2002.

[Fox 04] M. Fox, D. Long & K. Halsey. *An Investigation into the Expressive Power of PDDL2.1*. In European Conference on Artificial Intelligence, 2004.

[Ghallab 98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld & D. Wilkins. *Pddl - the planning domain definition language*. Rapport technique, 1998.

[Ghallab 04] M. Ghallab, D. Nau & P. Traverso. Automated planning theory and practice. Elsevier, 2004.

[Gottschlich 93] S. Gottschlich, C. Ramos & D. Lyons. *Assembly and Task Planning : A Taxonomy and Annotated Bibliography*. Rapport technique, 1993.

[Gravot 02] F. Gravot, R. Alami & T. Simeon. *Playing with several roadmaps to solve manipulation problems*. In Proc. IEEE IROS'02, 2002.

[Gravot 03a] F. Gravot & R. Alami. *An extension for handling multiple roadmaps and its use for complex manipulation planning*. In Proc. IEEE International Conference on Robotics and Automation, 2003.

[Gravot 03b] F. Gravot, S.Cambon & R.Alami. *aSyMov : a planner that deals with intericate symbolic and geometric problems*. Robotics Research, Tracts in Advanced Robotics, 2003.

[Gravot 04] Fabien Gravot. *aSyMov : Fondation d'un planificateur robotique intégrant le symbolique et le géométrique*. PhD thesis, Universite Paul Sabatier, Toulouse, 2004.

[Halperin 98] D. Halperin, J.-C. Latombe & R.H. Wilson. *A genreal framework for assembly planning : the motion space approach*. In 14th symposium on Computational Geometry, 1998.

[Hoffmann 03] J. Hoffmann. *The Metric-FF Planning System : Translating "Ignoring Delete Lists" to Numeric State Variables*. Journal of Artificial Intelligence Research, vol. 20, no. 20, pages 291–341, 2003.

[Hutchinson 90] S.A. Hutchinson & A.C. Kak. *A task planner for simultaneous fulfillment of operatotonal geometric and uncertainty-reduction goal*. AI Magazine, 1990.

[J.Cortés 02] J.Cortés, T.Siméon & J.-P. Laumond. *A Random Loop Generator for Planning the Motions of Closed Kinematic Chain using PRM Methods*. In Proc. IEEE International Conference on Robotics and Automation, 2002.

[J.Cortés 03] J.Cortés & T.Siméon. *Probabilistic Motion Planning for Parrallel Mechanisms*. In Proc. IEEE International Conference on Robotics and Automation, 2003.

[Kavraki 95] L.E. Kavraki, J.-C. Latombe, R. Motwani & P. Raghavan. *Randomized Query Processing in Robot Motion Planning*. 1995.

[Kavraki 96] L.E. Kavraki & P Svestka. *Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces*. IEEE Transactions on Robotics and Automation, 1996.

[Kuffner 98] J. Kuffner. *Motion Planning with Dynamics*. Physical, 1998.

[Laborie 96] P. Laborie. *IxTeT un système de planification hiérarchique gérant le temps et les ressources*. In RFIA, 1996.

[Lamare 98] B. Lamare & M. Ghallab. *Integrating a temporal planner with a path planner for a mobile robot*. In AIPS Workshop Integrating planning, scheduling and execution in dynamic and uncertain environments, 1998.

[Latombe 91] J.-C. Latombe. Robot motion planning. Kluwer Academic Publishers, Darien, CT, 1991.

[Laugier 85] C. Laugier & J. Troccaz. *SHARP, a system for automatic programming of manipulation robots*. Robotics Research : The third International Symposium, 1985.

[Laumond 94] J.-P. Laumond, P.E. Jacobs, M. Taix & R.M. Murray. *A Motion Planner for Nonholonomic Mobile Robot*. In Proc. IEEE International Conference on Robotics and Automation, 1994.

[LaValle 98] S. M. LaValle. *Rapidly-Exploring Random Trees : a New Tools for Path Planning*. Rapport technique TR98-11, Computer Science Dpt, Iowa State University, 1998.

[Leven 00] P. Leven & S. Hutchinson. *Toward Real-Time Path Planning in Changing Environments*. In Proc. of the Workshop on Algorithmic Foundations of Robotics (WAFR'00), 2000.

[Long 03] D. Long & M. Fox. *PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains*. Journal of Artificial Intelligence Research, vol. 20, no. 20, pages 61–124, 2003.

[Lozano-Perez 87] T. Lozano-Perez, J.L. Jones & E. Mazer. *Handey : A Robot System That Recognizes, Plans and Manipulates*. In ICRA, 1987.

[Lozano-Pérez 83] T. Lozano-Pérez. *Spatial Planning : A Configuration Space Approach*. In IEEE Transaction on Computers, 1983.

[Lynch 96] K. M. Lynch & M. T. Mason. *Stable pushing : Mechanics, controllability, and planning*. International Journal of Robotics Research, 1996.

[Mazon 90] I. Mazon, R. Alami & P. Violero. *Automatic Planning of Pick and Place Operations in presence of uncertainties*. In IROS'90, 1990.

[Nau 03] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu & F.Yaman. *SHOP2 : An HTN Planning System*. JAIR, 2003.

[Nissoux 99] C. Nissoux. *Visibilité et Méthodes Probabilistes pour la Planification de Mouvement en Robotique.* PhD thesis, Universite Paul Sabatier, Toulouse, 1999.

[O'Donnell 89] P.A. O'Donnell & T. Lozano-Perez. *Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators*. In IEEE Robotics and Automation Conference 1989, 1989.

[Pecora 04] F. Pecora, R.Rasconi & A.Cesta. *Assessing the Bias of Classical Planning Strategies on Makespan-Optimizing Scheduling*. In Proc. European Conference on Artificial Intelligence, 2004.

[Peng 03] J. Peng & S. Akella. *Coordinating the Motions of Multiple Robots with Kinodynamic Constraints*. In IEEE Robotics and Automation Conference, 2003.

[Qutub 98] S.W. Qutub. *Un paradigme générique pour la Navigation Coopérative de Robots Mobiles Autonomes*. PhD thesis, Universite Paul Sabatier, Toulouse, 1998.

[Reif 79] J.H. Reif. *Complexity of the Mover's Problem and Generalizations*. In Proc. 20th Symposium on Foundations of Computer Science, 1979.

[Régnier 91] Pierre Régnier & Bernard Fade. *Complete Determination of Parallel Actions and Temporal Optimization in Linear Plans of Action.* In Proc. European Workshop on Planning, 1991.

[Sahbani 03] Anis Sahbani. *Planification de tâches de manipulation en robotique par des approches probabilistes*. PhD thesis, Universite Paul Sabatier, Toulouse, 2003.

[Siméon 01] T. Siméon, J.-P. Laumond & F. Lamiraux. *Move3D : a generic platform for path planning*. 4th International Symposium on Assembly and Task Planning, 2001.

[Siméon 02] T. Siméon, S. Leroy & J.-P. Laumond. *Path Coordination for Multiple Mobile Robots : a resolution complete algorithms*. In IEEE Trasactions on Robotics and Automation volume 18, number 1, 2002.

[Siméon 03] T. Siméon, J.-P. Laumond, J. Cortés & A. Sahabani. *Manipulation Planning with Probabilistic Roadmaps*. International Journal Robotic Research, vol. 20, no. 20, pages 291–341, 2003.

[Stilman 04] Michael Stilman & James Kuffner. *Navigation Among Movable Obstacles : Real-time Reasoning in Complex Environments*. In Proceedings of the 2004 IEEE International Conference on Humanoid Robotics (Humanoids'04), December 2004.

[Svestka 97] P. Svestka. *Robot Motion Planning using Probabilistic Roadmaps*. PhD thesis, Universteit Utrecht, 1997.