N° d'ordre : 376 Année 2003

THÈSE

présentée en vue de
l'obtention du titre de

DOCTEUR
de
L'ÉCOLE NATIONALE SUPÉRIEURE
DE L'AÉRONAUTIQUE ET DE L'ESPACE

ÉCOLE DOCTORALE : SYSTÈMES

SPÉCIALITÉ : Systèmes industriels

par

Yannick MEILLER

Du morcellement de l'espace de recherche en planification d'actions

Soutenue le 28 novembre 2004 devant le jury :

MM.	L.	SIKLÓSSY	Président - Rapporteur
	C.	BARROUIL	
	P.	FABIANI	Directeur de thèse
Mme	M.	FOX	
MM.	F.	GARCIA	
	E.	JACOPIN	Rapporteur

Thèse préparée au sein du Département Commande des Systèmes et Dynamique du Vol de l'ONERA- centre de Toulouse

Thèse

présentée en vue de l'obtention du titre de

Docteur
de
L'Ecole Nationale Supérieure de l'Aéronautique et de l'Espace

Ecole Doctorale Systèmes : Systèmes Industriels, Outils Décisionnels

par

Yannick Meiller

Du Morcellement de l'espace de recherche en planification d'actions

Manuscrit

Remerciements

Le développement d'une thèse de doctorat est long. Il dure plusieurs années. Comme tout projet de recherche, il est caractérisé par une forte incertitude... avec son cortège de doutes, de belles surprises, de déconvenues, de motivation, d'intuition, de rêve et de stress aussi. Par ailleurs, le doctorat est une des rares formations où l'étudiant apprend sur l'ouvrage : il n'existe pas de "cours de recherche". Gérer l'incertitude s'y apprend donc aussi sur le tas.

Ainsi, un doctorat est avant tout une aventure humaine, faite de rencontres. Avec soi, forcément. Avec les autres : en plusieurs années j'ai rencontré beaucoup de gens! Il me faut les remercier tous et toutes car la préparation d'une thèse de doctorat prend une place importante dans la vie quotidienne. Elle interragit avec tout le reste. Je ne peux écrire autant de mercis ici, alors je vais me contenter de souligner quelques contributions clefs.

Commençons par le concret : la fin. Ce manuscrit existe et il a été soutenu devant un jury. Merci aux deux rapporteurs de ce jury : Messieurs Eric Jacopin et Laurent Siklòssy. Vous avez consacré beaucoup de temps à la lecture de ce manuscrit. Vous m'avez prodigué d'utiles conseils. Surtout, vous m'avez soutenu au moment difficile où il faut bien mettre un point final à "l'histoire sans fin". Je vous remercie aussi, ainsi que Madame Maria Fox et Messieurs Frédéric Garcia et Claude Barrouil, d'avoir accepté de participer au jury.

J'ai été accueilli pour ces travaux par le Département Commande des Systèmes et Dynamique du vol de l'Office National d'Etudes et de Recherches Aérospatiales. L'accueil, c'est des locaux, du matériel, un financement et encore une fois surtout des gens. Il y a bien sûr l'unité de recherche Conduite et Décision qui m'a beaucoup aidé. Il y a l'équipe administrative qui a toujours su démêler l'inextricable pour maintenir un environnement de travail efficace et agréable. Enfin, il y a la formidable ambiance de ce département et son soutien, même en des temps parfois difficiles. Pour cet accueil, je remercie les deux directeurs de département qui se sont succédés : Messieurs Jean-Pierre Jung et Claude Barrouil.

Je tiens aussi à remercier Monsieur Patrick Fabiani, mon directeur de thèse. La relation entre le doctorant et son directeur de thèse **fait** la thèse. Un directeur de thèse doit guider le doctorant tout en le laissant créer et s'exprimer. Il doit l'encourager et savoir lui faire abandonner une piste qui ne peut être explorée plus avant. Et puis... c'est une relation étroite qui dure plusieurs années! Nous avons très bien fonctionné ensemble. Merci pour tout, Patrick. Merci de ta compréhension et de ton soutien, pour la thèse et pour le reste.

Parmi tous les gens qui ignorent peut-être avoir contribué d'une manière ou d'une autre à ce que ce manuscrit existe, merci à Marc, Adrien, Michaël et Jean-Michel...

Enfin, mes parents. Je ne sais pas souvent le dire, alors voilà : merci.

Table des matières

Introduction

Intelligence Artificielle

Cette thèse s'inscrit dans le domaine de **l'Intelligence Artificielle**. Ce champ d'investigations, dénommé ainsi depuis 1956 environ, est particulièrement excitant : comprendre l'intelligence, maîtriser l'intelligence, concevoir des systèmes artificiels intelligents... Il renvoie à tout l'imaginaire exploité par les ouvrages et films de science fiction.

Il n'en reste pas moins que cet intitulé - "Intelligence Artificielle" - est vague. Qu'est-ce que l'intelligence? Comment peut-elle être évaluée? Est-ce un concept universel dont les intelligences animales sont des instances? En ce cas ces intelligences pouraient être des sources d'inspiration pour guider nos travaux. Au contraire l'intelligence doit-elle se concevoir uniquement comme humaine (voire animale)? Dans ce cas, ces intelligences animales seraient le sujet de nos travaux dont le but serait de les reproduire. Il serait hors de propos d'entrer ici dans ce débat. Le lecteur intéressé trouvera dans l'introduction de [68], une description plus détaillées des différents points de vue et un ensemble de références pour approfondir le sujet.

Pour introduire les travaux présentés ici, nous nous bornerons à une définition moins ambitieuse - tout à la fois non polémique et suffisante - de l'Intelligence Artificielle en Informatique. Nous rassemblons dans le domaine de l'Intelligence Artificielle tous les travaux qui visent à introduire plus d'autonomie dans les systèmes informatiques.

Ainsi, plutôt que d'indiquer à un système informatique comment mettre en œuvre une solution connue à un problème, l'idée est de lui indiquer comment trouver une solution face à un problème d'une certaine classe (et idéalement face à un problème quelconque). Le système devient alors moins dépendant de l'informaticien qui écrit ses programmes puisqu'au lieu d'un propramme par problème, il lui suffit d'avoir un seul programme pour de nombreux problèmes - voire tous les problèmes, dont certains pour lesquels l'informaticien ne connaît pas nécessairement de solution.

Dans ce but, l'Intelligence Artificielle comporte deux grands domaines. L'un concerne l'évaluation d'une situation. Il s'agit de reconnaître la situation courante à partir des données à disposition. C'est indispensable pour savoir quel est le problème à résoudre. L'autre s'attache à choisir une décision appropriée au problème courant. Bien évidemment ces deux domaines sont interdépendents puisqu'il est impossible de décider quoi que ce soit sans les bonnes informations, et que la valeur des informations dépend justement du processus de décision.

Nos travaux s'inscrivent dans la **partie décisionnelle de l'Intelligence Artificielle**.

Intelligence Artificielle décisionnelle et Planification d'actions

Les aspects décisionnels de l'Intelligence Artificielle cherchent à répondre à la question suivante : compte tenu de la description de la situation courante, que faut-il faire ? Cette interrogation peut s'appliquer à tous les domaines d'application. Voici quelques exemples :

1. Quelle direction choisir pour aller à tel endroit ?
2. Que faire si la température de tel ou tel réservoir dépasse une certaine valeur dans une installation de production chimique?
3. Comment une société doit-elle procéder pour acheter une série d'actifs tout en respectant certaines contraintes (ordre des achats, financement par emprunt bancaire, par augmentation de capital, par utilisation de fonds propres existants, ou par paiement en actions, etc.) ?
4. Comment charger ou décharger un camion de déménagement ?
5. Que faire pour obtenir un gâteau à partir des ingrédients disponibles?

La **planification d'actions** - notre domaine spécifique - est généralement essentielle pour répondre à ce type de questions. En effet, même si la question ne semble concerner que l'action à exécuter immédiatement, elle nécessite bien souvent un certain degré d'anticipation. La réponse donnée est alors la première étape d'une **séquence d'actions** (c'est-à-dire un **plan**) adaptée au problème. Considérons une à une les questions listées précédemment comme exemples :

1. "Aller à gauche" - parce que c'est là que se trouve la station de métro, qui va nous permettre de rejoindre la bonne zone de la ville, après quoi il suffira de descendre le boulevard, notre destination étant dans la deuxième rue à droite ;
2. "Lancer la procédure d'arrêt des ateliers de production en aval du réservoir trop chaud..." - parce que priver soudainement ces chaînes de production d'un réactif donné peut s'avérer plus dangereux que la hausse de la température en cours - "... ou isoler le réservoir s'il y a un risque d'explosion imminent (dépendant des valeurs de pression par exemple)" ;
3. Toute décision majeure d'une société s'inscrit dans une stratégie - c'est-à-dire un plan d'action ou une règle d'action conditionnelle ;
4. S'il y a des cartons de formes et de poids différents, il est sans doute préférable de placer les plus encombrants et les plus lourds en premier ;
5. Il n'existe pas de recette qui transforme un paquet de farine en gâteau en une seule action. Toute action entreprise l'est donc parce qu'elle permet d'amorcer une recette particulière. D'ailleurs, la recette elle-même est un plan d'actions.

Ainsi, qu'il s'agisse de produire explicitement un plan ou qu'il s'agisse plutôt de choisir la prochaine action à exécuter et de pouvoir argumenter ce choix, la planification d'actions entre en jeu. Cette forte présence de la planification d'actions dans les processus décisionnels l'a placée au centre de bien des intérêts de recherche et développement, dans des domaines tels que la robotique, la gestion de grands systèmes, l'économie, la logistique, les mathématiques, la préparation de missions militaires, etc.

Des chercheurs se sont intéressés au choix d'actions dans un ensemble et à leur mise en ordre pour apporter une solution à un problème donné. Ils se sont ensuite intéressés à des cas où l'ensemble d'actions possibles est de plus en plus grand, ou bien le problème particulièrement complexe. D'autres se sont plutôt attachés à garantir que le plan trouvé soit le meilleur possible - quitte à ne considérer que des problèmes simples dans un premier temps. Certains se sont focalisés sur la prise en compte d'informations incomplètes ou erronées. La problématique de la représentation des problèmes à traiter - son expressivité, sa facilité de mise en œuvre - est aussi un sujet d'étude.

Cette multiplicité d'angles d'étude a conduit au développement de plusieurs approches différentes. Chacune d'entre elles apporte un éclairage partiel sur la planification. La maturité actuelle du domaine mène au rapprochement de certaines d'entre elles afin de tirer parti des acquis qui leurs sont propres pour cumuler les atouts d'un planificateur tout en atténuant ses défauts.

Première partie : Planification d'actions, problématique et représentation

La première partie de ce mémoire de thèse est consacrée à la présentation des approches de la planification d'actions. Le premier chapitre présente plus précisément ce qu'est un problème de planification d'actions. Il introduit du vocabulaire et des notions propres à la planification qui seront utilisés dans la suite du texte.

Les trois chapitres suivant présentent trois grandes familles d'approches :

- Les premières approches présentées (chapitre 2) mènent leurs recherches d'une solution dans l'espace des états. En un mot, elles travaillent sur les conséquences des plans pour chercher une solution au problème posé.
- Le chapitre 3 présente des approches qui manipulent directement les plans eux-mêmes. Elles mènent leurs recherches dans l'espace des plans.
- Enfin, les approches présentées dans le chapitre 4 sont dites *disjonctives* car elles manipulent des disjonctions d'états (ou des disjonctions de plans).

Chaque chapitre présente en premier lieu les différentes approches en se plaçant dans le cadre du *problème classique de planification* (tel que défini dans le chapitre 1). Dans un second temps, les aspects concernant l'optimisation et le traitement des incertitudes sont présentés. Ceci permet de comparer aisément les approches entre elles.

En particulier, cet état de l'art vise à mettre en lumière les différentes façons dont les espaces de recherches sont structurés et en particulier comment les informations sont agrégées pour être traitées. Une synthèse est proposée qui structure notre approche du problème dans cette thèse

Seconde partie : notre contribution

Notre thèse est qu'une part essentielle des spécificités de chaque approche provient de la structuration de son espace de recherche. Plus précisément, elle provient de ce qu'elle distingue et de ce qu'elle abstrait. La seconde partie de ce mémoire de thèse présente notre argumentation.

Dans le premier chapitre, nous introduisons la notion de *morcellement de l'espace de recherche* comme caractéristique fondamentale d'une approche de planification. Adoptant une démarche constructrice plutôt que descriptive, nous montrons qu'en modifiant simplement la manière de *morceler* l'espace de recherche, nous pouvons obtenir un degré d'arborescence plus ou moins grand - reproduisant tout aussi bien le comportement d'approches disjonctives que celui d'approches travaillant dans l'espace d'états. Ce faisant, nous donnons une base unificatrice aux nombreuses approches que nous avons évoquées, facilitant ainsi les hybridations des unes avec les autres.

Afin de mettre en œuvre notre théorie, nous avons développé le planificateur `TokenPlan`. Ce planificateur permet de mettre en pratique de nombreuses stratégies de morcellement, définissant ainsi autant d'approches différentes de la planification. Sa présentation, dans le chapitre 7, permet d'avoir une vision concrète du morcellement de l'espace de recherche et de la manière dont ce dernier spécifie une approche de planification.

Le chapitre suivant revient précisément sur la notion de morcellement de l'espace de recherche. Il montre en quoi une stratégie de morcellement est liée aux objectifs d'une approche de planification : rapidité, optimisation, traitement de l'incertitude, etc. En cela nous retrouvons les spécificités qui étaient naturellement apparues entre les différentes approches présentées en partie 1. Ce faisant, nous montrons aussi que le morcellement de l'espace de recherche n'introduit pas seulement de l'arborescence dans l'espace de recherche et qu'il est plus expressif encore.

Enfin, le dernier chapitre de la seconde partie présente quatre séries d'expérimentations réalisées avec `TokenPlan`. Elles montrent comment la définition d'une stratégie de morcellement permet tout autant de reproduire le comportement de planificateurs existants que d'obtenir des comportements nouveaux. En particulier, nous nous attardons sur la mise en œuvre de techniques de type *Branch and Bound* dans un contexte de planification disjonctive, à des fins d'optimisation.

D'une part, ces expérimentations apportent des éléments de validation de l'expressivité descriptive de la notion de morcellement de l'espace de recherche dans un rôle unificateur des approches existantes - en reproduisant certains comportements existants. D'autre part, elles montrent l'adéquation de cette notion avec une démarche constructrice, au travers des résultats en matière "d'optimisation disjonctive". En cela, elles ouvrent de nombreuses perspectives quant au développement de nouvelles approches de planification, en particulier dans le cadre du traitement de l'optimisation et de l'incertitude.

Première partie

Planification

Chapitre 1

Planification d'actions : problématique et représentation

La planification d'actions est une fonction décisionnelle. Au sein d'un système de prise de décisions, elle permet d'aller au delà du choix de la prochaine action à exécuter en fonction de l'état courant. En effet, procédant par anticipation et analyse des futurs possibles, elle choisit un ensemble d'actions à exécuter en plusieurs étapes. Chaque choix est ainsi intégré dans un plan d'actions cohérent, bâti lors de la planification pour atteindre les objectifs en fonction des connaissances disponibles à ce moment-là.

Le rôle d'un planificateur est d'être capable de générer automatiquement des plans d'actions adaptés à des problèmes différents, sans nécessiter de reprogrammation "ad hoc" pour chaque instance. La diversité des problèmes qu'un même planificateur peut traiter diffère selon les approches de planification mises en œuvre. Certaines se focalisent sur un type de problèmes spécifique et recherchent essentiellement l'efficacité sur ce type de problèmes. C'est le cas de la planification de mouvements par exemple. Le lecteur intéressé pourra se reporter à l'annexe B. Dans le cadre de nos travaux, nous nous intéressons au contraire aux approches les plus génériques. En effet, elles mettent en avant de manière plus évidente les enjeux et les difficultés liées à la planification d'actions.

L'objet de ce chapitre est de poser les fondements de la planification. Qu'est-ce qu'un problème de planification? Comment se représente-t-il? Comment est-il étendu pour permettre de traiter les problèmes plus réalistes?

1.1 Le problème de planification

La planification d'actions est aujourd'hui un domaine de recherche très large qui traite d'une grande variété de problèmes. Cependant, elle a été fondée autour d'un type de problèmes particulier formulé à la fin des années soixante dans la communauté de *l'Intelligence Artificielle*. Comme repère, citons par exemple [23].

Pour ces raisons historiques, ce type de problèmes et les travaux s'y rapportant ont été qualifiés de *classiques* : il est question du *problème classique de planification* et de la *planification classique*. Il est important de noter que ce type de problèmes n'est ni le plus courant dans les applications réelles, ni le plus général au niveau théorique, ni le seul à être étudié. Malgré tout, le terme "planification classique" est toujours communément utilisé (par exemple dans [1], [35], ou encore dans le tutorial présenté par S. Kambhampati lors de la conférence

AAAI'2000[1]). Des synthèses de la planification classique peuvent être trouvées dans [68] (chapitres 11 à 13), [72], [50], ou encore [36].

Problème classique de planification - formulation informelle : A partir d'une situation initiale donnée, et compte tenu de ce qu'il est possible de faire, quel enchaînement d'actions exécuter pour atteindre une situation souhaitée.

Par exemple, le problème suivant est un problème de planification.
Je suis à mon domicile (situation initiale), *Je souhaite aller à l'aéroport* (situation finale souhaitée - les objectifs). *Que faire ?* Notons qu'ici ce qu'il est possible de faire est implicite. En effet, en tant qu'humain normalement constitué, lister tout ce que je peux faire est fastidieux.

Une réponse possible peut-être le plan suivant :
Préparer les bagages. Prendre la voiture. La conduire jusqu'au parking de l'aéroport. Entrer dans l'aéroport.
Le plan suivant est une autre réponse qui semble tout aussi valide : *Préparer les bagages. Retirer de l'argent. Appeler un taxi. Lui indiquer d'aller à l'aéroport. Le payer. Entrer dans l'aéroport.*

Bien-sûr, dans le cadre de la planification automatique d'actions, il n'y a pas de place pour l'implicite. Tous les éléments indispensables à la résolution du problème doivent être décrits au planificateur. Ils sont au nombre de quatre.

Eléments d'un problème de planification: Un problème de planification est constitué de quatre composants:

1. Le système contrôlé. Il est aussi question *d'agent* plutôt que de système - nous ne ferons pas de différences entre ces termes tout au long de ce document. Il s'agit du système dont les actions sont planifiées. Dans l'exemple, il s'agissait de *Je*.
2. L'environnement. Il s'agit en fait de tout ce qui n'est pas le système contrôlé. Dans l'exemple, c'est aussi bien la valise que le taxi - entre autres. En particulier, il est important de savoir comment l'agent peut interagir avec son environnement (remplir la valise, se déplacer en utilisant un taxi...)
3. La situation initiale.
4. Les objectifs à atteindre.

L'exemple tel qu'il est formulé est incomplet - nous l'avons noté - car rien n'est indiqué à propos de l'agent, ni à propos de son environnement. Ces données sont primordiales pour un planificateur. D'une part, elles définissent le champ des possibilités dans lequel le raisonnement peut se construire. Par exemple, le planificateur ne peut pas proposer d'utiliser la voiture de l'agent s'il n'est pas mentionné que l'agent en a une, ni qu'une voiture permet sous certaines contraintes de changer de position au sein de l'environnement géographique.

D'autre part, elles constituent le vocabulaire du problème. La situation initiale, les objectifs, et la solution ne peuvent s'exprimer qu'en des termes qui proviennent de la description de l'agent et de son environnement. Prenons l'exemple d'un agent évoluant dans une ville. Si les positions et les déplacements dans la ville sont décrits en termes de stations de métro, de rues, de places, de bus, et de voiture, alors décrire la position de l'agent dans la situation initiale en termes de longitude et de latitude est sans aucun sens. Cela place la situation initiale en dehors du monde.

1. "Recent Advances in AI Planning: a Unified View", support de présentation disponible à l'URL suivante : http://rakaposhi.eas.asu.edu/planning-tutorial/

Il existe différents types d'environnement et d'interaction entre l'agent et l'environnement. Dans le cadre du problème classique de planification, l'environnement est *statique*.

Définition : *Environnement statique vs. dynamique*
Un environnement *statique* ne change que sous l'effet des actions de l'agent. Un environnement *dynamique* comporte des objets *non contrôlés* pouvant évoluer indépendamment des décisions de l'agent.

Par exemple, un agent jouant au Solitaire est dans un environnement statique, alors que s'il descend une rivière en canoë il est dans un environnement dynamique. En effet, dans ce dernier cas, même si l'agent ne fait rien, l'environnement change.

Par ailleurs, dans le problème classique, toutes les actions de l'agent sont *déterministes*.

Définition : *actions déterministes vs. non déterministes*
Dans le cas d'une *action déterministe*, son exécution dans un contexte donné ne peut produire qu'un seul ensemble de conséquences. Cet ensemble est connu pour chaque contexte d'exécution possible. Au contraire, une *action non déterministe* peut avoir différents ensembles de conséquences sans qu'il soit possible de savoir a priori lequel va effectivement apparaître.

Ces deux limitations du problème classique - environnement statique et actions déterministes - permettent de fusionner les descriptions de l'agent et de l'environnement pour n'en garder que les interactions. L'agent se réduit à ses capacités d'action. L'environnement se réduit à son comportement face à chaque action possible de l'agent. Cette description est ce que l'on appelle le *domaine de planification*. Par exemple, [51] est un langage dédié à la description de domaines de planification.

Définition : *domaine de planification*
Un *domaine de planification* décrit tout ce qu'il est possible de faire pour le système contrôlé. Il définit toutes les actions possibles du système contrôlé sur son environnement.

Cette notion de domaine est importante pour l'objectif de généricité des planificateurs. En effet, un même planificateur pourra résoudre des problèmes de robotique mobile ou de stratégie financière en exploitant dans chaque cas un domaine différent. Les spécificités de chaque champ d'application sont réunies dans leurs domaines de planification respectifs.

Définition : *problème classique de planification*
Un problème classique de planification est entièrement défini (cf. par exemple [36], [72] ou [50]) par :

1. un domaine de planification
2. une situation initiale
3. des objectifs à atteindre

Le domaine de planification décrit un environnement statique où les actions de l'agent sont déterministes.

Pour le lecteur plus habitué aux travaux de la communauté satisfaction de contraintes, il est important de noter une petite difficulté de vocabulaire liée au terme *problème*. En effet, *domaine* et *problème* en planification d'actions correspondent respectivement à *problème* et *instance* en satisfaction de contraintes...

Les solutions que génère un planificateur (s'il en trouve) sont des *plans*.

Définition : *Plan solution du problème classique*

Deux éléments composent un plan solution du problème classique :

1. un ensemble d'actions sélectionnées parmi celles définies dans le domaine de planification du problème ;
2. un ordre sur les actions de cet ensemble déterminant de quelle manière elles doivent s'enchaîner.

L'enchaînement d'actions ainsi défini doit respecter deux contraintes :

1. son exécution peut débuter dans l'état initial ;
2. d'après le modèle de l'agent et de l'environnement défini par le domaine de planification, à l'issue de son exécution l'agent a atteint ses objectifs.

1.2 Représentation

Le problème de planification classique est à présent clairement défini. Reste à pouvoir formuler ses éléments de telle manière qu'ils soient exploitables par un planificateur. C'est le rôle de la *représentation*.

En planification, la notion fondamentale de la représentation des choses est *l'état*. Les différentes représentations se divisent en deux catégories selon qu'elles y font référence de manière explicite ou implicite.

1.2.1 Notions d'état et de changement d'état

La planification d'actions gère et exploite le *changement*. En effet, si un plan d'action est recherché afin d'atteindre des objectifs, c'est que ces objectifs ne sont pas déjà atteints à l'instant considéré. La situation courante doit être *modifiée* de telle manière que la nouvelle situation renferme les objectifs.

Cela implique que ces modifications puissent être décrites. Il faut pouvoir raisonner sur ce qui *est* avant la modification et ce qui *est* après. C'est le rôle des *états* ([62]).

Définition : *état*

Un *état* est un résumé de ce que sont l'agent et l'environnement à un instant donné. L'objectif de décrire des *états* est de pouvoir en étudier les différences et de pouvoir étudier les modifications permettant de passer de l'un à l'autre.

Cette notion d'état existe dans le langage commun : il est question par exemple des *états comptables d'une société*, ou de *l'état de santé* d'une personne.

Considérer les états commes des repères dans le cours du changement est essentiel dans le cadre de la planification d'action. C'est ce qui va diriger la manière de les bâtir. En effet, une même situation peut être décrite par des états différents selon les aspects de la situation qu'ils soulignent. Un état étant un *résumé*, des choix sont nécessairement effectués quant à ce qu'il inclut et ce qu'il laisse de côté.

Une analogie peut être faite avec les positions intermédiaires d'un itinéraire. Si un agent doit aller de Toulouse à Lyon en train, il suffit de distinguer les gares de Toulouse et de Lyon. Pour le même parcours en voiture, il convient de distinguer plusieurs états intermédiaires qui correspondent aux changements d'autoroutes, aux sorties d'autoroutes, etc. .

De la même manière, concernant la position d'un agent, plusieurs niveaux de description sont possibles :

1. il est à Paris ;
2. il est à Paris, dans le quinzième arrondissement, près de la station de métro Charles Michels ;
3. il est dans le salon ;
4. il est sur une chaise.

Si l'agent doit se rendre à l'arc de Triomphe, à Paris, les descriptions 1, 3 et 4 ne sont pas adaptées. En effet, seule la deuxième localise l'agent à l'intérieur de Paris. En revanche, cette localisation précise dans la ville est inutile si l'agent organise son voyage autour du monde en avion.

C'est le domaine de planification qui définit le degré de finesse des états. Celui-ci doit être adapté au type de problèmes à traiter. Un domaine mal écrit peut empêcher la résolution des problèmes soumis. Au mieux il la rendra plus difficile en fournissant trop de détails inutiles.

Le problème classique de planification se reformule en termes d'états.

Définition : *problème classique de planification en termes d'états*

Un problème classique de planification est entièrement défini par :

1. un domaine de planification ;
2. un état initial ;
3. un ensemble d'états qui doit être atteint.

Le domaine de planification décrit un environnement statique où les actions de l'agent sont déterministes.

La figure 1.1 représente graphiquement un exemple de problème de planification. L'agent concerné est un robot mobile - *Robby*. Il peut se déplacer de salle en salle. Il a deux pinces lui permettant de saisir des objets. En particulier il peut transporter des balles. En l'occurrence, dans l'état initial, il se situe dans la salle 1, ses deux pinces libres. Trois balles sont avec lui. Le but visé est d'avoir les trois balles dans la salle 2. Dans cet exemple, que Robby soit ou non dans la salle 2 à l'issue du plan n'a pas d'importance. Les objectifs définissent un ensemble d'états et non un état unique.

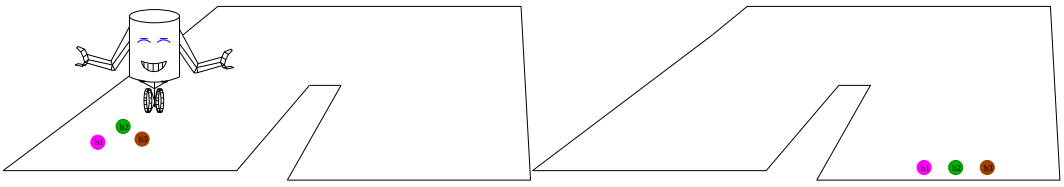


FIG. 1.1 – *Voici un exemple de problème de planification. Le schémas de gauche représente l'état initial. Le schéma de droite représente les objectifs à atteindre. Il ne s'agit pas d'un état en particulier car la position du robot Robby n'est pas stipulée, il peut donc être dans n'importe quelle pièce. En outre, dans cet exemple, la position particulière des balles à l'intérieur d'une pièce n'est pas considérée.*

1.2.2 Représentation en extension

Il existe une manière très simple de représenter les actions et les états d'un système. Il suffit de lister à l'avance tous les états possibles, ainsi que les transitions possibles qui les relient. Il est alors question de représentation *en extension*.

En général, il est fait usage de représentations matricielles. Une *matrice de transitions* rassemble toutes les transitions possibles du système.

Définition : *matrice de transitions*

Il s'agit d'une matrice carrée dont les lignes, comme les colonnes, sont indexées par les numéros d'états. La valeur de $M_{i,j}$ désigne l'action qui lorsqu'elle est exécutée dans l'état i mène à l'état j ($\emptyset$ si cette transition n'existe pas) (cf. figure 1.2).

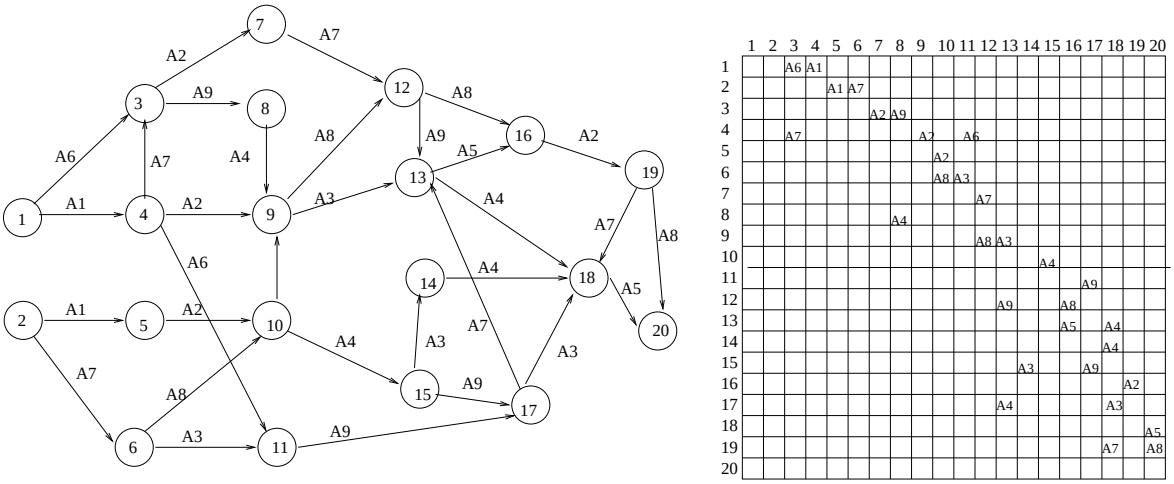


FIG. 1.2 – *Le schémas de gauche représente un ensemble d'états et les actions qui les relient entre eux. La matrice de transitions correspondante est représentée à droite. Pour alléger la matrice les cases contenant le symbole $\emptyset$ ont été laissées vide.*

1.2.3 Représentation en intension

Les représentations utilisées en planification symbolique sont toutes dérivées du langage utilisé dans le système STRIPS [23], introduit en 1971, et en restent assez proches. En 1998, le langage PDDL [51] a été créé dans un souci d'uniformisation des formalismes afin de pouvoir échanger aisément entre équipes des problèmes tests. Il permet de transcrire la plupart des variantes de STRIPS. Il a été complété depuis. Une deuxième version a été publié fin 2001 ([25]).

Une grande part des travaux en planification repose sur la manipulation de symboles, ce qui lui permet d'utiliser une représentation en intension. Ainsi, tous les états possibles ne sont pas eux-mêmes listés. Seules leurs caractéristiques, ou propriétés, le sont. Par combinaison de celles-ci, il est possible de reconstituer les états. La figure 1.3 présente un exemple simple. Dans la suite de ce mémoire, lorsqu'il sera question de représentation en intension, sans autre précision, nous nous réfèrerons au type spécifique de représentation présenté ici, c'est-à-dire fondé sur la description des état par un ensemble de propriétés (et inspiré par le langage de STRIPS).

Nous considérons un robot mobile qui peut se déplacer de salle en salle. Une description de domaine pourrait retenir les caractéristiques suivantes :

- (Sont-connectées $salle_i$ $salle_j$)
- (Est-dans *salle robot*)
- (Position-bras élévation robot) - ici l'élévation peut être basse ou haute (quand le robot lève les bras au dessus de sa tête - dans cet exemple les deux bras bougent toujours ensemble)

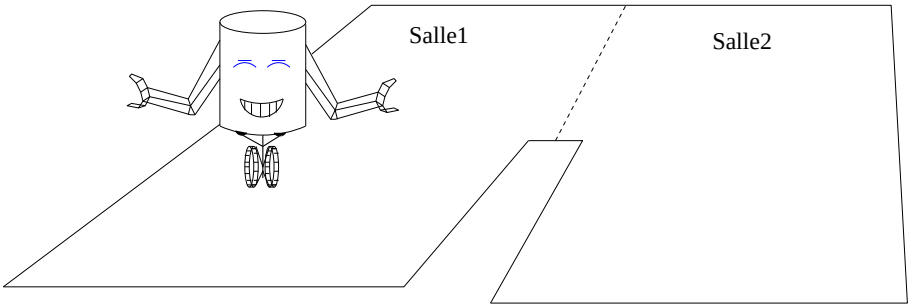


FIG. 1.3 – *Avec le domaine que nous avons décrit, cet état se représente par {(Sont-connectées salle1 salle2) (Est-dans salle1 robby) (Position-bras basse Robby)}*

Définition : *prédicat*

Un *prédicat* désigne la description d'une caractéristique (ou propriété) en termes de variables - par exemple, (Est-dans *salle robot*). La variable *salle* peut être instanciée per exemple par *salle*1, et la variable *robot* par *Robby*.

Définition : *proposition*

Une *proposition* désigne un prédicat totalement instancié - par exemple, (Est-dans *salle*1 *Robby*). La variable *salle* est instanciée par *salle*1, et la variable *robot* par *Robby*.

Définition : *opérateur*

Un opérateur définit une transition possible entre deux ensembles d'états par deux ensembles de propriétés :

- un ensemble de préconditions ;
- un ensemble d'effets.

Un opérateur peut être exécuté dans tout état où toutes les préconditions sont valides. Les effets déterminent les changements apportés à l'état - à savoir les caractéristiques à ajouter (parfois appelés *effets positifs*), et celles à enlever (parfois appelés *effets négatifs*).

Notre exemple pourrait contenir l'opérateur (Aller robot, $salle_i$, $salle_j$), avec pour préconditions (Est-dans $salle_i$ robot) (Sont-connectées $salle_i$, $salle_j$), pour seul effet positif (Est-dans $salle_j$ robot), et pour seul effet négatif non(Est-dans $salle_i$ robot).

Définition : *action*

C'est un opérateur totalement instancié, c'est-à-dire un opérateur dont toute les variables ont une valeur définie. Par exemple, (*Aller Robby*, *cuisine*, *salon*) est une action (fondée sur l'opérateur (*Aller robot*, $salle_i$, $salle_j$)).

Il est important de noter qu'en combinant les caractéristiques il est possible de définir des **ensembles** d'états. Par exemple, la caractéristique (Position-bras basse Robby), considérée indépendemment, définit un ensemble d'états contenant celui de la figure 1.3 mais aussi celui où Robby est dans l'autre salle, les bras baissés. En particulier, les préconditions d'une action ne définissent pas un état précis mais l'ensemble des états dans lesquels cette action peut être exécutée.

La représentation en extension est une représentation sans structure ([11]). Toutes les connaissances du domaine et du problème ont été utilisées pour construire les tables d'états et de transitions. Ensuite ce sont ces tables qui sont utilisées pour la planification.

Au contraire, la représentation en intension que nous avons décrite ici conserve la structure du domaine. En effet, plutôt que de savoir simplement que l'action $A1$ mène de l'état 28 à l'état 73 (cas de la représentation en extension), nous pouvons savoir ici ce que modifie cette action $A1$ dans l'état dans lequel elle est exécutée. Nous pouvons connaître quelles propriétés de cet état fait que $A1$ peut y être exécutée.

A des fins exclusivement explicatives, le lecteur peut faire un parallèle avec une base de données relationnelle simple ([13, 17]). Les propriétés définissent les relations. Les propositions correspondent aux tuples de chacunes de ces relations. Ainsi, à un instant donné, l'ensemble de la base de données contient toutes les caractéristiques valides dans l'état correspondant. Les opérateurs indiquent comment les tables correspondant aux relations peuvent être modifiées en fonction de ce qu'elles contiennent. Les actions correspondent à des modifications précises. L'exécution d'une action provoque une transition entre états, et parallèlement modifie la base de données. Ainsi une action se réduit à une opération de mise à jour de la base de données, par suppression (effet négatif) ou ajout (effet positif) de tuples. De la même manière que des erreurs de mise à jour peuvent aboutir à une base de données incohérente, un domaine mal écrit peut mener le planificateur à générer des plans invalides dans la réalité. L'écriture d'un domaine doit donc être réalisée avec soin et attention.

1.3 Extensions du problème classique

Le problème classique de planification est trop restrictif pour permettre de traiter tous les problèmes réalistes. Nous décrivons ici trois extensions qui permettent de considérer de nombreux problèmes qui ne peuvent l'être dans le cadre du problème classique.

1.3.1 Objectifs non-terminaux

En planification, deux types de tâches peuvent être distinguées :

1. atteindre un but terminal ;
2. assurer un service.

Le premier type de tâches est couvert par le problème classique de planification. En revanche, ce n'est pas le cas du second. En effet, assurer un service nécessite de la part de l'agent de **se maintenir** dans un ensemble d'états spécifiés plutôt que de **l'atteindre**.

Ainsi, `Rejoindre un lieu` est clairement un problème d'atteinte de but terminal, alors que `suivre une personne dans ses déplacements` n'en est pas un. Dans le dernier cas, le but est d'assurer un service, le service étant de garantir une distance inférieure à un seuil spécifié entre la personne et l'agent.

Une solution pour traiter une tâche du second type est de la "discrétiser" en une suite (potentiellement infinie) d'atteintes de buts terminaux. Ces dernières sont autant de problèmes classiques qui peuvent être traités en séquence. C'est l'approche utilisée dans [20] par exemple.

Si la mission peut être bornée dans le temps (assurer tel service pendant une heure par exemple), il peut être envisagé de résoudre à l'avance l'ensemble du problème, en prenant en compte toutes les situations pouvant apparaître. Si la mission n'a pas de durée pré-déterminée mais qu'elle a un caractère cyclique, cette démarche peut être appliquée à un cycle.

Enfin, une piste plus radicale est de revoir totalement la notion de tâche et d'en tirer les conséquences sur l'architecture des agents. C'est dans cet esprit que Stein introduit la notion de *système interactif* dans [69]. C'est aussi part intégrante d'une réflexion plus globale concernant l'informatique ambiante (en anglais : *pervasive computing*) [15], qui traite du développement effectif de l'omniprésence de l'informatique au travers de l'intégration et de l'interconnexion de processeurs au sein de plus en plus d'objets de la vie quotidienne.

1.3.2 Qualité des solutions

Quand bien même les objectifs concernent l'atteinte d'un ensemble d'états, les problèmes réalistes soulèvent souvent la question de la **qualité** du plan. Il ne suffit pas d'atteindre les objectifs, il faut aussi que le plan correspondant soit le plus rapide, consomme le moins d'énergie ou encore utilise le moins possible telle action, par exemple. L'optimisation du plan rend la planification beaucoup plus difficile que la planification classique. En effet, afin d'extraire le meilleur plan - au sens des critères prédéfinis - il est nécessaire soit d'évaluer explicitement **tous** les plans existants, soit de pouvoir garantir qu'il n'existe pas de meilleur plan parmi ceux qui ne sont pas évalués explicitement.

1.3.3 Prise en compte de l'incertitude

Le plus souvent, les problèmes réalistes de planification ne permettent pas une modélisation exhaustive et exacte de l'environnement. La cause peut en être une méconnaissance du système. C'est par exemple le cas lorsqu'il s'agit de planifier un parcours à partir d'une carte incomplète ou qui n'est pas à jour.

Elle peut aussi être un choix : quand il est jugé plus raisonnable de s'accommoder d'une certaine imprécision plutôt que de déployer tous les efforts scientifiques et techniques nécessaires à la modélisation exacte du système et à l'exploitation d'un tel modèle ensuite.

Une source courante des limites de la modélisation de l'environnement est la présence d'un environnement dynamique. En effet, dans ce cas l'exécution des actions n'est pas la seule cause de modification de l'état du système. En conséquence, l'approche du domaine de planification, qui est de décrire l'environnement et sa dynamique au travers seulement des actions possibles de l'agent, devient vite trop limitée.

Par exemple, un agent poursuivant un autre agent, ou bien un problème de planification d'une production de céréales - de la semence à la récolte - concernent des environnements dynamiques. Clairement, les mouvements de la cible et la météorologie sont des facteurs importants du problème mais il ne sont pas contrôlés par les actions à disposition du planificateur.

Cette source n'est cependant pas la seule. Un environnement statique peut générer un modèle d'interaction avec l'agent de grande complexité. Par exemple, les roues d'un robot mobile peuvent patiner plus ou moins selon la nature du sol, la pente, les conditions météorologiques etc. . Généralement, aucun modèle n'est utilisé pour prédire avec exactitude ce degré de patinage. En conséquence, mettre les roues en rotation à n tours par minute pendant x minutes peut déplacer l'agent de distances plus ou moins grandes. Si de plus, les roues n'ont pas toutes patiné de la même manière, le robot a pu changer d'orientation!

En planification, deux types d'incertitude sont pris en compte :

- incertitude sur les effets des actions (actions non déterministes)
- incertitude quant à l'état courant effectif

Dans le cas d'une action non déterministe, ses effets ne sont pas prévus avec précision. Cela signifie que lorsqu'elle est exécutée dans un état donné, il n'est pas possible de savoir quel état est atteint. Le plus souvent un ensemble d'états candidats est connu.

C'est le cas par exemple lorsqu'un automobiliste décide de prendre le périphérique d'une ville importante. Si la circulation y est fluide, il arrivera à destination plus rapidement qu'en traversant le centre-ville. Par contre, si un accident est survenu, il est probable qu'un embouteillage se forme et il mettra alors bien plus de temps que s'il avait traversé le centre-ville. Cependant il ne le sait pas a priori.

Dans le cadre des agents autonomes, c'est aussi le cas, par exemple, d'un robot dont les roues patinent plus ou moins en fonction du terrain, ou encore dont la pince peut ne pas réussir à saisir un objet parce qu'il est humide et glissant.

Des actions non-déterministes peuvent aussi être utilisées pour traiter un environnement dynamique. Dans ce cas, les modifications non-controlées de l'état peuvent être incluses comme effets incertains des actions. Par exemple, dans le cas de la poursuite d'une cible mobile, l'action de ne rien faire peut aboutir à autant d'états différents qu'il y a de possibilités pour la cible de se déplacer. Cette approche permet de considérer l'environnement comme statique. Sa dynamique est tranférée sur le non déterminisme des actions.

L'autre forme d'incertitude concerne la connaissance pendant l'exécution de l'état dans lequel se trouve l'agent. On parle *d'observabilité* ([62]).

Définition : *observabilité*

Il est question *d'observabilité totale* quand l'état courant peut être intégralement connu. Dans le cas contraire, il est question *d'observabilité partielle*, voire *d'inobservabilité.*

Lorsque l'agent sait exactement dans quel état il se trouve, l'observabilité est totale. Notons que cela n'est pas incompatible avec des actions incertaines. En effet, dans notre précédent exemple, au moment d'entrer sur le périphérique, l'automobiliste sait où il se trouve et l'heure qu'il est. Il ne sait pas a priori à quelle heure il atteindra la sortie visée - du fait du non déterminisme de l'action. Par contre, quand finalement il sort du périphérique, là encore il sait où il se trouve et l'heure qu'il est.

En revanche, lorsqu'un patient arrive à l'hôpital, son état n'est pas connu avec précision. Bien-sûr, les symptômes visibles et la gamme d'examens disponibles permettent de cerner la

situation. L'observation est *partielle*. Notons cette fois, que ceci n'est pas incompatible avec des actions déterministes. En effet, supposons être sur les Champs Elysées à Paris. C'est facile à reconnaître, donc nous sommes sûrs de notre position. Par contre nous ne savons pas dans quel sens nous marchons - nous allons soit vers l'est, soit vers l'ouest. L'action avancer à partir d'une position sur les Champs Elysées, en faisant face à l'ouest, rapproche de manière certaine de l'arche de la Défense. A partir de la même position en faisant face à l'est, elle rapproche de manière certaine de la place de la Concorde. Ne sachant pas quelle est notre orientation initiale, il nous est impossible d'en savoir plus après avoir avancé de quelques mètres. Peut-être plus loin rencontrerons nous un nouvel indice permettant de resserrer encore le champ des possibilités - une station de métro ou une enseigne célèbre par exemple.

Considérons maintenant un robot sans aucune capacité perceptive. L'observabilité est alors *nulle*.

1.4 Extensions de la représentation : valeur et incertitudes

Pour trouver le meilleur plan, il faut pouvoir évaluer un plan, et ce même en présence d'incertitudes. De nombreuses approches de manipulation de données incertaines existent. [18] présente un état de l'art complet à ce sujet. Néanmoins, le cadre formel le plus utilisé pour ce faire est la théorie des jeux [49].

Le concept clef de la théorie des jeux est la notion *d'utilité*.

Définition : *fonction d'utilité*

La *fonction d'utilité* associe à chaque état une valeur numérique. Cette fonction transcrit une préférence sur les états. Cette préférence est déterminée par les objectifs et les contraintes imposées au système.

Notons l'importance de la notion d'état dans cette définition. En effet, la théorie des jeux est fondée sur une représentation en extension des états. En conséquence, cette fonction d'utilité n'est pas facilement adaptable à la représentation en intension de la planification classique. En effet, puisqu'une propriété individuelle définit un ensemble d'états, il n'est pas facile de lui attribuer une valeur d'utilité unique.

Dans le cas déterministe, planifier revient alors à trouver la séquence d'actions aboutissant à la séquence d'états la plus utile, c'est-à-dire dont la somme des utilités élémentaires récoltées est la plus élevée.

En présence d'incertitudes, une même séquence d'actions peut mener à différentes séquences d'états, et donc différentes utilités globales. Il est alors nécessaire d'aggréger les utilités de ces divers chemins possibles (cf. figure 1.4). Localement, l'utilité d'un état dépend de l'utilité des états accessibles en y exécutant l'action prévue par le plan. L'utilité de ces états est elle-même calculée de la même manière. Le procédé est donc récursif.

Plusieurs critères sont possibles pour obtenir une valeur d'utilité en fonction des utilités des issues possibles de l'exécution d'une action :

- comportement pessimiste : l'utilité minimale est retenue. Il s'agit d'une étude au pire cas. L'agent ne prend pas de risque, mais peut rater les meilleures opportunités.
- comportement optimiste : l'utilité maximale est retenue. L'agent tente le tout pour le tout...
- comportement "raisonnable" : l'espérance mathématique des utilités est retenue - c'est-à-dire leur moyenne pondérée par leurs probabilités respectives d'occurrence. Ce critère

est souvent utilisé. Il a cependant plusieurs défauts. En particulier une même moyenne peut correspondre à des extrêmes bien différentes. Par exemple, peut-être êtes-vous prêt à éventuellement perdre dix euros dans l'espoir de gagner un million d'euros... mais seriez-vous pour autant prêt à jouer dix mille euros pour les mêmes chances de gagner cette fois un milliard d'euros? L'espérance mathématique des deux cas peut être rigoureusement identique (avec une probabilité de gain de $1/100001$)[2]...

Par ailleurs, plus l'horizon de planification est lointain, moins ce critère est discriminant. En effet, le nombre de possibilités croissant toutes les issues possibles se moyennent et au final se valent à peu près, donnant à l'agent un comportement hésitant.

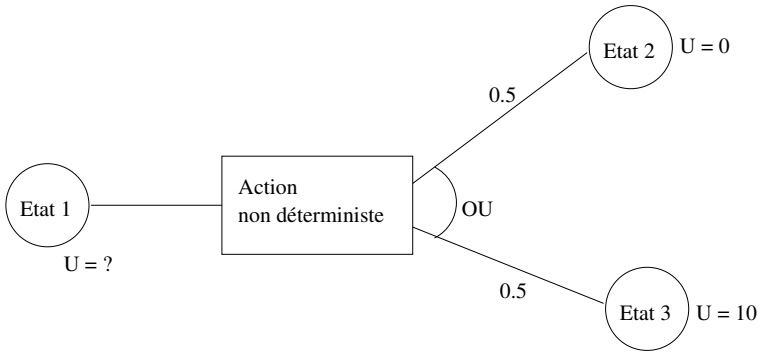


FIG. 1.4 – *Voici un exemple d'action non déterministe. Exécutée dans l'état 1, elle peut mener soit à l'état 2 avec une probabilité de 0.5, soit dans l'état 3 avec une probabilité de 0.5. L'état 2 est associé à une utilité de 0, alors que celle de l'état 3 est 10. Dans ce cas de figure, quelle est l'utilité de l'état 1? Dans le cadre d'un comportement pessimiste, elle est de 0 (seul le pire cas est retenu). Avec un comportement optimiste elle est de 10 (seul le meilleur cas est retenu). Avec un comportement "raisonnable", la moyenne pondérée par les probabilité d'occurence est calculée : 5.*

Dans ce cadre, la représentation est en extension. Chaque état a une utilité intrinsèque. Dans le cas incertain, une matrice de transition est définie pour chaque action. Cette fois, $M_{i,j}$ contient la probabilité que l'exécution de l'action considérée dans l'état i mène à l'état j.

1.5 A problèmes étendus, solutions étendues : classification des divers types de plans

Le plan défini pour répondre au problème classique de planification n'est pas adapté pour les extensions de ce problème. En premier lieu, l'observation de l'environnement doit pouvoir être prise en compte.

Face à l'incertitude, trois types de plans peuvent apporter une réponse complémentaire : le plan *conformant*, le plan *contingent*, et le plan optimisant un compromis succès/incertitudes.

2. Cas de loteries parfaitement équilibrée pour lesquelles l'espérance de gain est de 0.

1.5.1 Boucle ouverte vs. boucle fermée

Considérant l'interaction entre l'agent et son environnement, l'image d'une boucle est souvent utilisée. La première moitié de la boucle représente l'action de l'agent sur l'environnement. La seconde moitié transcrit le flux d'information remontant de l'environnement vers l'agent.

Quand ce *feedback* n'est pas présent, il est question de *boucle ouverte*. S'il est présent, il est question de boucle fermée (cf. figure 1.5).

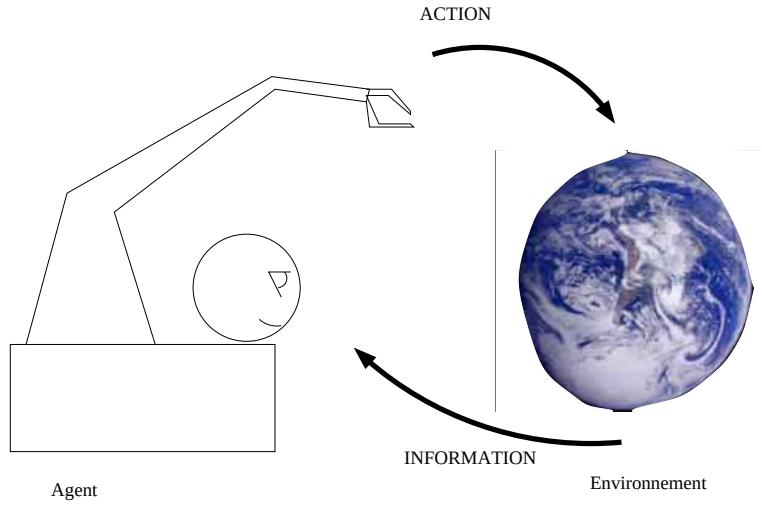


FIG. 1.5 – *L'agent agit sur son environnement. S'il peut observer l'environnement, la boucle d'interaction entre l'agent et son environnement est fermée. Il peut adapter son action suivant l'information qu'il reçoit.*

Définition : *plan en boucle ouverte*

Un plan en boucle ouverte ne comporte que des actions qui ont une influence sur l'environnement. Par exemple, une solution à un problème classique de planification est un plan en boucle ouverte.

Un plan *en boucle ouverte* suppose que le modèle utilisé est parfait puisqu'aucune vérification de l'état courant réellement atteint ne sera effectuée pendant l'exécution.

Définition : *plan en boucle fermée*

Un plan en boucle fermée contient des observations de l'environnement et prend en compte le résultat de ces observations.

1.5.2 Plan conformant

Un plan *conformant* est la réponse le plus généralement souhaitée pour un problème incertain. Un plan de ce type garantit que son exécution mène aux objectifs, quoi qu'il arrive parmi les aléas recensés dans le domaine de planification (cf. figure 1.6)

Définition : *plan conformant*

Un *plan conformant* est un plan en boucle ouverte qui garantit le succès de la mission de l'agent quels que soient les aléas pris en compte au moment de la planification.

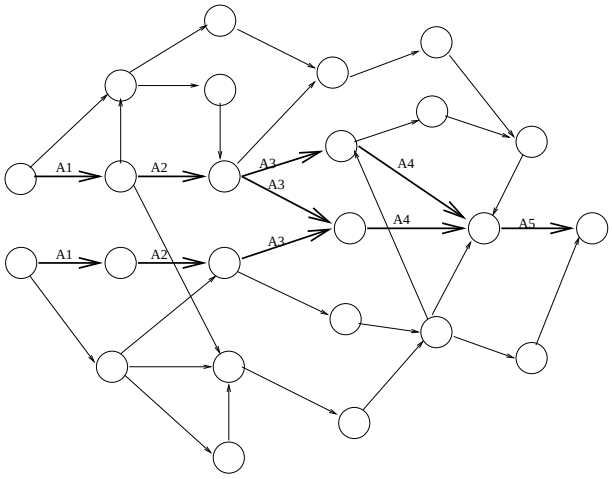


FIG. 1.6 – *Chaque cercle représente un état différent. Les deux états les plus à gauche sont les deux états initiaux possibles (incertitude sur l'état initial). L'état le plus à droite est un état objectif. En gras figure un plan conformant. En effet, cette succession d'actions peut être exécutée à partir de n'importe lequel des deux états possibles elle aboutit toujours au but. Notons que A3 est même non déterministe dans un cas (à partir d'un même état elle atteint potentiellement deux états différents).*

Lors de la planification cela revient à toujours se dire "Compte tenu de telle ou telle incertitude nous ne pouvons pas savoir exactement dans quel état va nous mener telle action mais nous savons avec certitude qu'il appartiendra à tel ensemble". Cela est rarement possible.

1.5.3 Plan contingent

Une alternative au plan conformant qui garantit aussi le succès de la mission malgré les incertitudes est le plan *contingent*.

Définition : *plan contingent*

Un *plan contingent* est un plan en boucle fermée qui garantit le succès de la mission de l'agent quels que soient les aléas prévisibles au moment de la planification.

Ce type de plan nécessite de la part de l'agent des capacités d'observation, et un contrôle d'exécution permettant de les gérer. En effet, un plan contingent est fait de sous-plans conformants correspondant à différentes occurences des aléas recensés dans le domaine de planification. Il inclut aussi - aux points de jonction entre diverses possibilités - des instructions d'observation dont les résultats - lors de l'exécution - permettront de choisir entre telle ou telle branche conformante (cf. fig. 1.7).

Bien-sûr, toute information n'est pas disponible à tout instant. De plus, observer peut avoir un coût dont il faut tenir compte - en particulier cela peut empêcher d'agir. Supposons par exemple qu'un agent mobile soit contraint à s'arrêter pour filmer son environnement, pour des questions de stabilité du système de prise de vue. Il lui est alors impossible d'observer en permanence pour être certain de choisir la bonne route, car dans ce cas il ne pourrait de toute façon pas bouger! Ainsi, la planification contingente soulève le problème de la planification

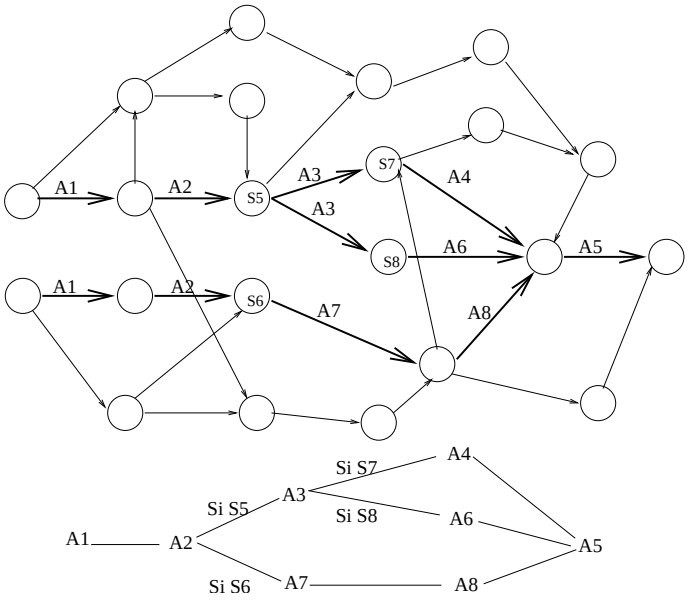


FIG. 1.7 – *Dans le schéma du haut, chaque cercle représente un état différent. Les transitions retenues dans le plan contingent sont en gras. Le schéma du bas représente le plan contingent extrait. Il montre des portions en boucle ouverte (conformantes) et des branchements. Au niveau de chaque branchement, il faut connaître l'état courant effectif afin de choisir la bonne branche. La fin du plan est à nouveau conformante.*

d'observations, et plus généralement le problème du dilemme perception-action. Un exemple en robotique mobile détaille ce type de difficultés dans [22].

Notons quelques nuances de vocabulaire. Un plan *contingent* est un plan *conditionnel* puisque l'exécution de certaines "branches" est **conditionnée** aux résultats d'observations. En l'occurrence, ce conditionnement vise à faire face aux contingences liées au problème de planification à résoudre.

Une *politique*, parfois appelée *stratégie*, désigne une généralisation du plan contingent. Il s'agit d'une fonction qui pour tout état d'un ensemble pré-déterminé donne l'action à exécuter pour se rapprocher des objectifs. L'exécution de cette action assure que l'état suivant est toujours dans l'ensemble d'états traité par la politique. Ainsi, à partir de l'un quelconque de ces états, suivre les indications de la politique assure d'atteindre les objectifs.

Si une telle assurance n'est pas possible elle garantit au moins que le chemin suivi optimise les chances de succès. L'optimisation des chances de succès est l'objet du troisième type de plans.

1.5.4 Optimisation d'un compromis

Trouver un plan conformant ou un plan contingent est une tâche de satisfaction. Ces types de plans garantissant le succès, il suffit d'en trouver une instance pour arrêter la recherche. Malheureusement, de tels plans n'existent pas toujours.

La seule alternative est de maximiser les chances de succès. Le problème devient alors un problème d'optimisation, où le critère de qualité n'est autre que le degré d'incertitude - considéré comme un coût à minimiser (cf. par exemple [20]).

1.6 Conclusion

Le rôle et la place de la planification d'actions dans un système décisionnel sont aisés à comprendre. Ceci explique certainement que la communauté de l'Intelligence Artificielle se soit rapidement intéressée à la question. Plus intéressée par les mécanismes de la planification que par ses applications immédiates, elle s'est orientée vers une sorte d'idéal que constituent les planificateurs génériques.

Les travaux se focalisent pour l'essentiel sur la résolution d'un type de problèmes spécifique : le problème classique de planification. Ils constituent la planification classique. La partie de son héritage sur laquelle nous nous focalisons ici est double. D'une part, il y a la notion de domaine de planification - qui est à la base de la généricité des planificateurs. D'autre part, elle a mis en avant une représentation des états en intension basée sur des descriptions en termes de propriétés. Cette représentation permet de décrire de manière compacte et structurée des systèmes complexes. En outre, elle permet aussi de manipuler des ensembles d'états.

Malheureusement, le problème classique de planification a une expressivité limitée. Des notions communes dans les applications réelles telles que la valeur d'un plan, des contraintes à respecter, ou encore la nécessité de tenir compte d'incertitudes, ne peuvent être exprimées dans le cadre classique de la planification.

La représentation en intension, créée sur mesure pour le problème classique, se retrouve difficilement adaptable à ces extensions. En effet, les notions adéquates sont souvent héritées de la théorie des jeux laquelle se fonde sur une représentation en extension des états.

Chapitre 2

Planification dans l'espace des états

Chercher un plan dans l'espace des états est sans doute la démarche la plus intuitive. C'est d'ailleurs la première qui fut mise en œuvre. Il s'agit simplement de parcourir les états atteignables de proche en proche jusqu'à trouver un chemin continu reliant l'état initial à un état objectif. Une fois un tel chemin trouvé le plan se construit en extrayant les actions qui permettent de passer de chaque étape du chemin à la suivante.

2.1 FSS, BSS

Les premières approches en planification classique ont utilisé un parcours arborescent (cf. annexe A) de l'espace des états. La *recherche par chaînage avant dans l'espace des états* (*Forward State Space search* - *FSS* en Anglais), part de l'état initial.

Pour chaque action pouvant y être exécutée, une nouvelle branche est introduite. Elle mène à l'état résultant de l'application de ladite action. De proche en proche, se construit un arbre représentant l'atteignabilité des états à partir de l'état initial. Si une feuille correspond à un état objectif, alors un plan existe. Il correspond à la séquence d'actions se succédant sur le chemin reliant la racine à cette feuille. La figure 2.1 montre un exemple de cette approche sur un problème de transport de balles par Robby.

Une variante de cette approche est de partir des objectifs plutôt que de l'état initial. Il est alors question de *recherche par chaînage arrière dans l'espace des états* (*Backward State Space search* - *BSS* en Anglais). Le principe est le même mais dans ce cas les actions sont introduites si leurs effets sont compatibles avec les objectifs. Les branches mènent aux états d'où auraient pu être exécutées ces actions pour mener aux objectifs. La première action introduite est donc candidate pour être la dernière du plan (le plan est construit en partant de la fin). Lorsqu'une feuille contient l'état initial, la planification s'arrête. Le plan est le chemin reliant cette feuille à la racine.

L'avantage de cette approche sur *FSS* est de se focaliser sur les objectifs. En effet, ne sont pris en compte que les actions et les états pouvant mener au but. La figure 2.2 reprend l'exemple traité en chaînage avant pour le développer en arrière.

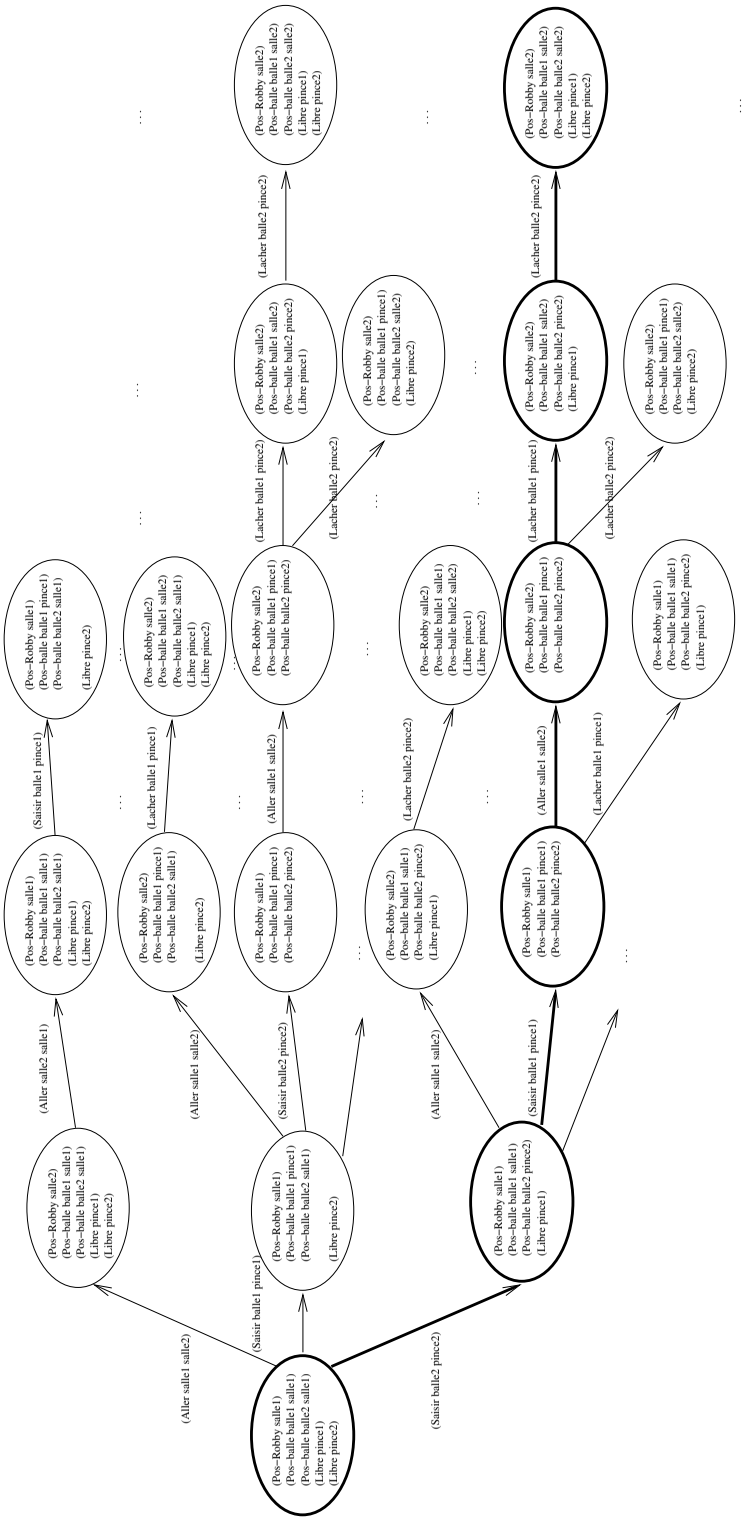


FIG. 2.1 – *Voici un arbre de recherche par chaînage avant (*FSS*) dans l'espace d'états. Il n'est représenté que partiellement par manque de place. Dans l'état initial Robby le robot est dans la salle 1, avec deux balles, ses deux pinces sont libres. L'objectif est d'avoir les deux balles dans la salle 2. Le chemin en gras correspond à un plan possible.*

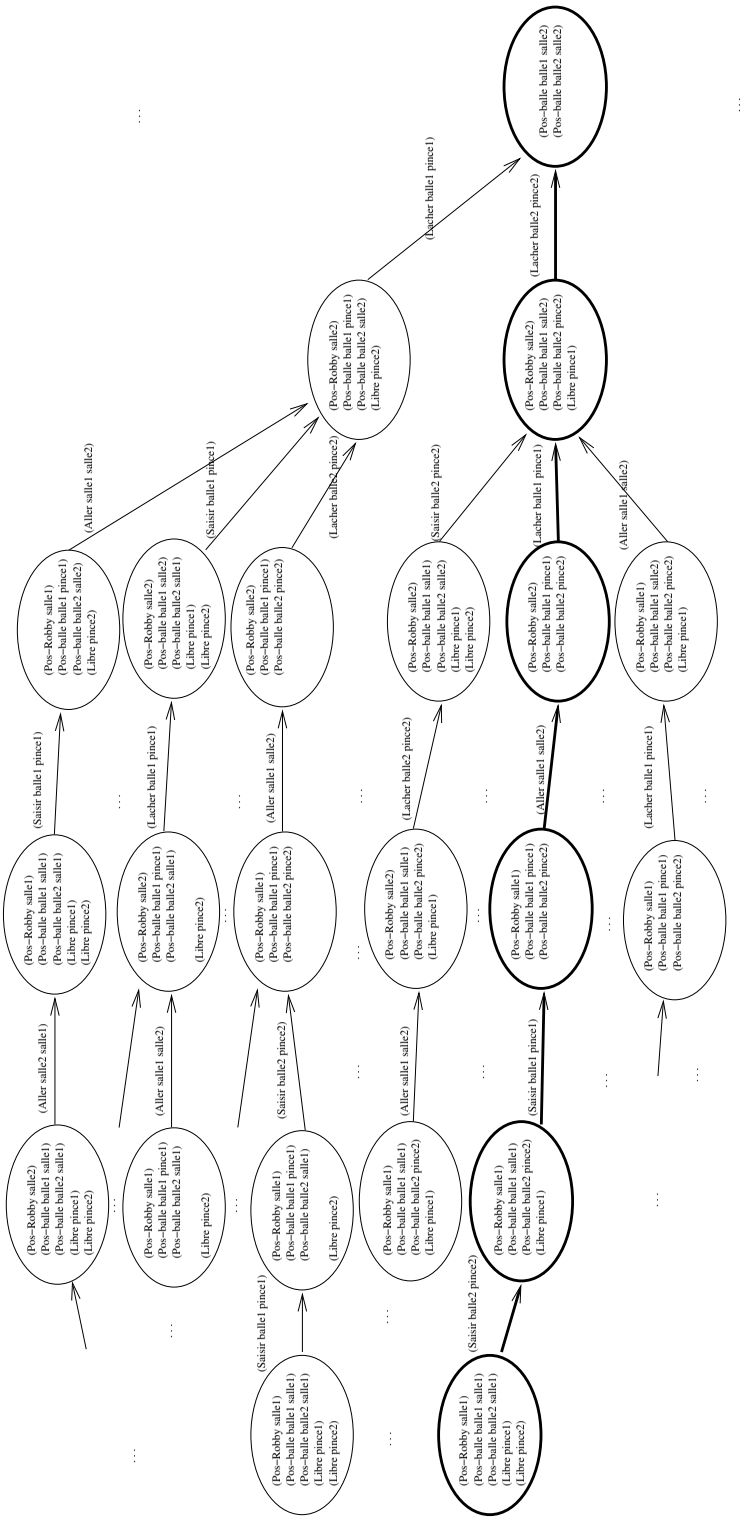


FIG. 2.2 – *Voici un arbre de recherche par chaînage arrière (*BSS*) dans l'espace d'états. Il n'est représenté que partiellement par manque de place. Le problème traité est le même que celui de la figure 2.1. Le même plan est mis en évidence ici.*

Notons que dans un cas comme dans l'autre, le parcours arborescent doit se faire *en largeur d'abord* (cf. annexe A). En effet, il existe des plans de longueur infinie. Dans notre exemple, le simple plan qui amène Robby à prendre une balle, à la reposer, puis à la reprendre et ainsi de suite, en est un.

Dans le cadre d'une recherche en *profondeur d'abord*, un tel plan correspondrait à une branche infinie. Si le planificateur explore une telle branche avant de trouver la solution, alors il ne la trouvera jamais car il ne passera jamais à une autre branche.

Par ailleurs, il est intéressant de constater que ces approches ne tirent pas parti des avantages de la représentation en intension qu'elles utilisent. En effet, elles manipulent les états individuellement. Avec une représentation en extension elles se comporteraient de la même manière puisqu'elle n'ont pas besoin de manipuler des ensembles d'états.

Ce type d'approches s'est révélé inefficace en planification du fait de la taille astronomique que peuvent atteindre ces arbres. En particulier, ils mènent à des duplications de travail inutiles. En effet, plusieurs nœuds peuvent correspondre au même état. Les sous-arbres dont ils sont les racines sont alors rigoureusement identiques. Développer des efforts pour chacun d'entre eux est de ce fait une perte de temps. La figure 2.3 montre l'exemple d'une telle duplication. Il est cependant possible de vérifier pour chaque nœud que l'état correspondant n'a pas déjà été atteint de façon à ne pas refaire le travail. Le surcoût à contrôler est alors celui causé par cette identification des états.

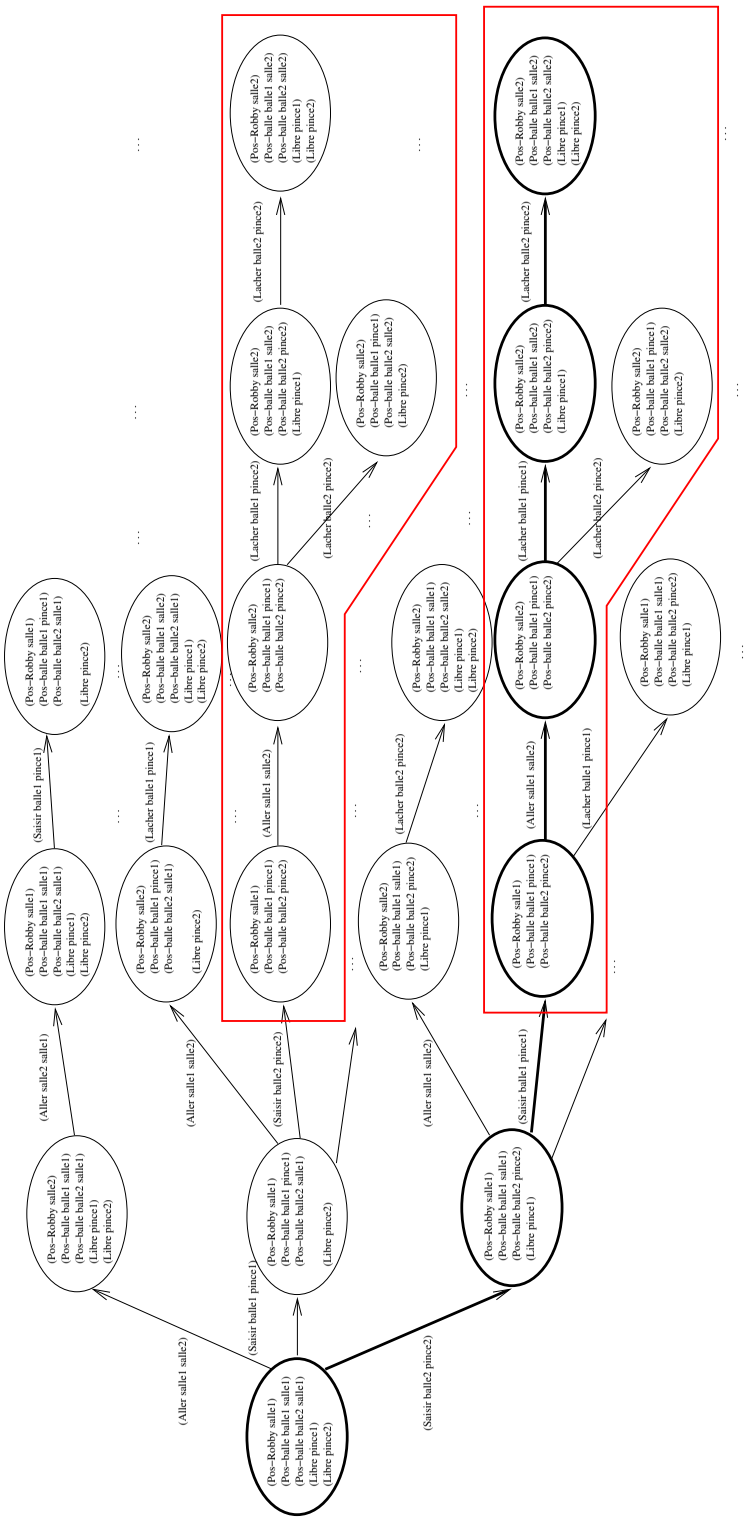


FIG. 2.3 – *Cette figure reprend l'arbre de recherche de la figure 2.1. Les deux sous arbres encadrés sont rigoureusement identiques. Ils le sont car leurs racines respectives correspondent à un même état découvert deux fois.*

2.2 Programmation dynamique

La programmation dynamique naquit dans les années cinquante, avec l'énoncé du *principe d'optimalité* ou *Principe de Bellman*[4]. Elle s'intègre dans le cadre de la résolution de problèmes d'optimisation séquentiels et peut s'appliquer à tous les domaines pour lesquels le Principe de Bellman est vrai ([67] montre que le Principe de Bellman ne s'applique pas toujours).

Principe de Bellman (R. Bellman 1949) : Toute politique optimale ne peut être formée que de sous-politiques optimales.

Ce principe déclare que toute partie d'un chemin optimal est elle-même optimale. Il a mené au développement d'approches basées sur l'énumération implicite des solutions. En n'énumérant pas explicitement les chemins - contrairement à *FSS* ou *BSS*, les problèmes de duplication inutile de recherches sont évités. Ceci a aussi pour conséquence de générer des politiques plutôt que des plans.

Les approches de programmation dynamique utilisent une représentation en extension de l'espace d'états (avec une matrice de transitions - cf. section 1.2.2, ou un graphe de transition).

A chaque état est associée une valeur du critère à optimiser. Différents algorithmes de résolution sont basés sur une mise à jour itérative de ces valeurs. A la fin du processus, la valeur associée à un état s est la valeur du chemin optimal entre s et un état objectif. Il est question de *programmation dynamique arrière*. Notre présentation va se borner à celle-ci. Cependant, il existe aussi la *programmation dynamique avant*. Dans ce cas, la valeur associée à un état s est la valeur du chemin optimal entre l'état initial et s. Les différences de mise en œuvre sont assez faibles pour que la présentation de la progammation dynamique arrière permette de comprendre facilement la programmation dynamique avant. Ces différences concernent essentiellement les équations de récurrence mettant en œuvre le principe d'optimalité.

Ainsi, la solution est obtenue sour forme de politique ou stratégie qui indique pour tout état l'action à exécuter. Il s'agit simplement de celle qui mène à l'état voisin associé à la meilleure valeur du critère (cf. figure 2.4). Pour récupérer le plan menant de l'état initial à un état objectif, il faut partir de l'état initial et suivre le chemin indiqué par les meilleures valeurs.

Dans le cadre de la résolution d'un problème classique de planification, l'optimisation se fait sur un *critère terminal*. Toutes les valeurs des états sont initialisées à une constante choisie a priori, sauf les états objectifs qui reçoivent une valeur correspondant à la récompense finale (fig. 2.5). Si la récompense finale est choisie suffisamment grande, alors il s'en suit qu'un plan optimal est simplement un plan menant à un état objectif - nous sommes revenus à un problème de satisfaction.

Un défaut des méthodes de programmation dynamique en arrière est de résoudre le problème considéré pour tout état initial dans l'espace de recherche. C'est ce qui permet de produire une politique optimale. Cependant, dans le cas où nous cherchons simplement un plan solution, l'espace entier va être fouillé, même si l'état initial et les objectifs sont proches [1]. La programmation dynamique avant présente un défaut similaire: le problème considéré est résolu pour tout état objectif, à partir d'un état initial fixé.

Les différentes approches se répartissent en deux grands types. Celles utilisant *l'itération de la valeur*, et celles utilisant *l'itération de la politique*.

1. Ceci est totalement vrai pour la programmation dynamique par itération sur la valeur standard. Il existe des algorithmes qui tentent de pallier ces difficultés avec plus ou moins de succés.

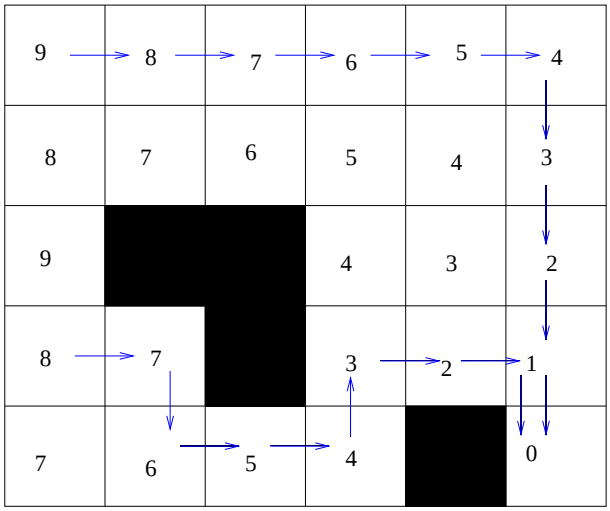


FIG. 2.4 – *Cette figure considère un problème de déplacement sur une grille 2D. Le but est d'aller dans la case en bas à droite en minimisant le nombre de déplacements élémentaires. Le résultat d'une approche programmation dynamique est représenté. Chaque valeur correspond au coût du chemin optimal entre cet état et l'objectif. Deux plans sont présentés. L'un a pour état initial le coin supérieur gauche, et l'autre une case plus au sud - toujours dans la colonne de gauche. Les actions dirigent toujours l'agent vers l'état ayant la plus petite valeur. Quel que soit l'état de départ, l'agent sait quoi faire pour atteindre le but de manière optimale - il s'agit bien d'une politique.*

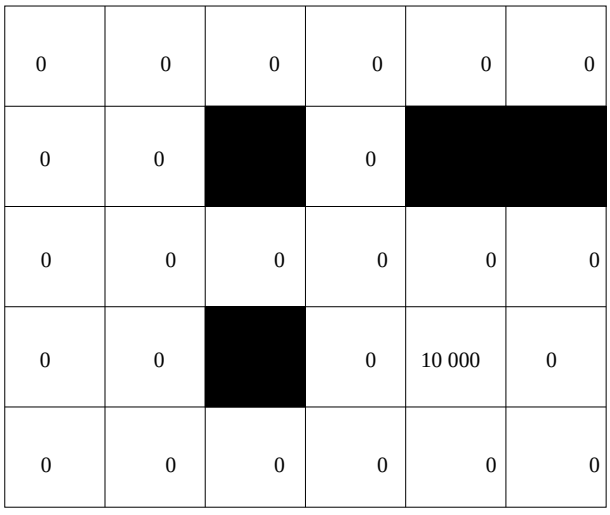


FIG. 2.5 – *Voici un exemple de critère terminal. L'état de valeur initiale strictement positive est l'état objectif. Pour revenir à un problème de satisfaction pure, aucun coût ou gain n'est associé aux transitions. Peu importe le chemin pour y parvenir, la récompense unique s'obtient en atteignant l'état objectif.*

2.2.1 Itération de la valeur

Choisissons s_2 tel qu'il soit le successeur immédiat de s_1. La valeur du chemin entre s_1 et s_2 est simplement le coût de l'action qui relie ces deux états. Si le chemin optimal entre s_2 et le but est connu, alors il suffit d'ajouter à sa valeur le coût de l'action entre s_1 et s_2 pour connaître la valeur du chemin global. Pour savoir s'il reste optimal, il suffit de faire de même avec tous les successeurs possibles de s_1 et de comparer.

C'est sur ce principe que se base la mise à jour des valeurs des états. Elles sont d'abord initialisées à 0, sauf pour les états objectifs qui ont pour valeur une récompense terminale constante (au début et par la suite). Ensuite, la valeur de tous les états (sauf les états objectifs) est mise à jour suivant les équations d'optimalité de Bellman (cf. [62]) :

$$V^k(s) = max_{a \in A}(c(s,a,s') + V^{k-1}(s')) \tag{2.1}$$

L'équation 2.1 est l'équation de récurrence à la base de l'itération de la valeur. A est l'ensemble des actions, $c(s,a,s')$ désigne le coût associé à l'exécution de l'action a dans l'état s (ce qui mène à l'état s'). $V^k(s)$ est la valeur de s lors de la mise à jour de l'itération k. $V^{k-1}(s)$ est la valeur de s à l'itération $k-1$.

Après la première itération (c'est-à-dire après le calcul des $V^1(s)$ pour tout s), les seuls états à changer de valeur sont ceux frontaliers avec un état objectif. De proche en proche - d'itération en itération - la vague de mise à jour se propage jusqu'à avoir touché tous les états (cf. figure 2.6). Il est important de noter qu'un état peut changer de valeur plusieurs fois. En effet, la mise à jour des valeurs d'états éloignés du but peut ouvrir de nouveaux chemins. Ceux-ci peuvent être de nouvelles alternatives à partir d'états déjà considérés et donc remettre en cause les valeurs déjà calculées (cf. figure 2.7).

Une fois que tous les chemins ont été explorés, les chemins optimaux ont nécessairement été trouvés. En conséquence, les choix d'action restent figés. Il s'en suit qu'aucune valeur n'est modifiée d'une itération à une autre. L'algorithme peut alors s'arrêter.

En général, la convergence est lente et l'algorithme est arrêté lorsque la valeur des $V^k(s)$ ne change pas plus qu'un ϵ d'erreur fixé.

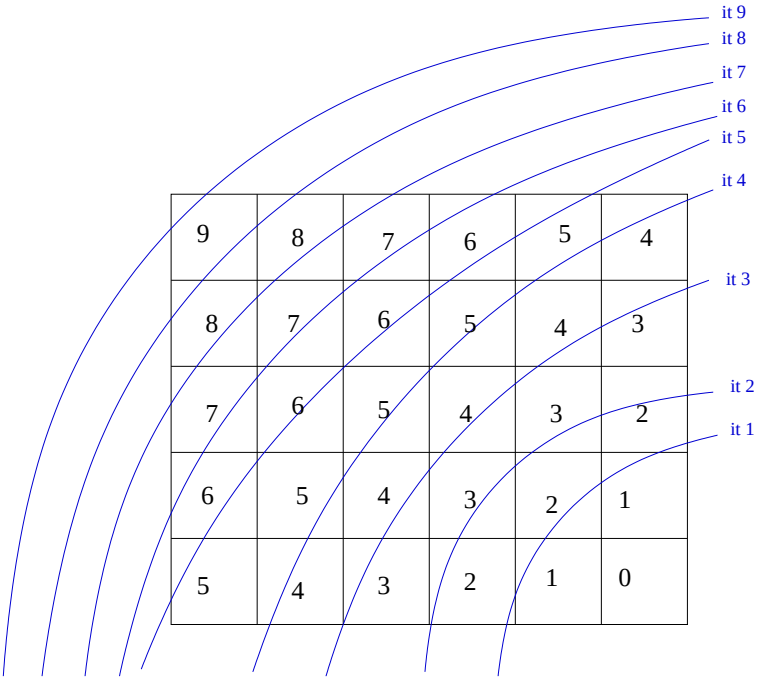


FIG. 2.6 – *En programmation dynamique arrière, les valeurs se propagent de proche en proche. Ici le but est de se rendre au coin inférieur droit en minimisant le nombre de déplacements. Il faut neuf itérations avant que le coin supérieur gauche ne voie sa valeur modifiée.*

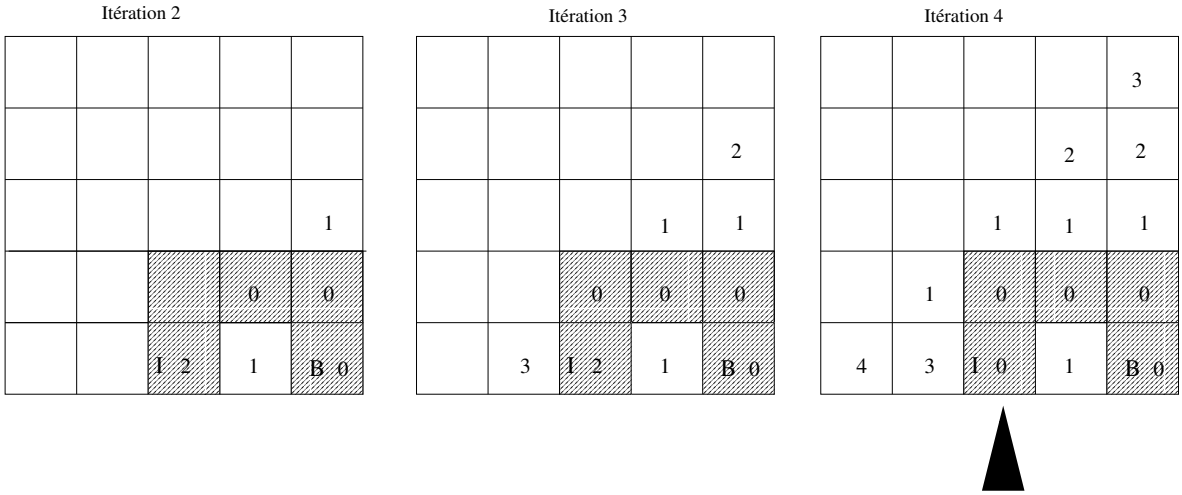


FIG. 2.7 – *Voici trois itérations consécutives d'une résolution par itération de la valeur. Le problème posé est de se rendre de l'état I à l'état B en minimisant un coût. Ce coût est de 1 pour chaque déplacement d'une case à l'autre, sauf entre cases hachurées - où il est nul. Nous voyons que lors de la quatrième itération, une nouvelle diagonale de valeurs est mise à jour (propagation de proche en proche), mais de plus la valeur de l'état I - déjà calculée lors de l'itération 2 - est modifiée.*

2.2.2 Itération de la politique

A cause des délais de propagation, la liste des valeurs des états peut continuer à varier alors que la politique sous-jacente reste stable.L'*itération de la politique* apporte une réponse à ce problème. L'idée est de mettre à jour la **politique** à chaque itération plutôt que les valeurs des états qui en découlent.

Ainsi, pour chaque état est retenue l'action à exécuter pour suivre le plan optimal jusqu'au but. L'équation 2.1 devient :

$$P^k(s) = argmax_{a \in A}(c(s,a,s') + V^{P^{k-1}}(s')) \tag{2.2}$$

L'équation 2.2 est l'équation de récurrence à la base de l'itération de la politique. A est l'ensemble des actions, $c(s,a,s')$ désigne le coût associé à l'exécution de l'action a dans l'état s (ce qui mène à l'état s'). $V^{P^{k-1}}(s')$ est la valeur de s' lorsque la politique définie lors de l'itération précédente est exécutée.

Après chaque itération il faut calculer la valeur de chaque état en appliquant la politique courante. Cette étape est coûteuse. Il s'agit de la résolution d'un système linéaire à n équations - n étant le nombre d'états [2]. En contre-partie la convergence vers une politique optimale est généralement plus rapide.

2.3 Approches heuristiques

Un moyen de pallier les difficultés des approches arborescentes est d'utiliser des heuristiques pour les guider. Il s'agit de fonctions qui indiquent des préférences sur la manière de développer l'arbre. Elles indiquent en particulier les branches les plus prometteuses en termes de satisfaction des objectifs et de nombre de coups pour y arriver. Des algorithmes génériques de parcours arborescent aidés par une heuristiques ont été développés de longue date (cf. [56] et annexe A).

Dans le cadre de la planification, les travaux se sont donc portés essentiellement sur la construction automatique de l'heuristique. L'idée est d'utiliser la description du domaine de planification pour générer automatiquement une heuristique adaptée au problème. Cette notion de génération automatique et adaptée est importante car nous sommes dans le cadre de la planification générique.

En effet, l'usage d'heuristique n'est pas récent. Cependant, les heuristiques étaient définies par le concepteur, sur mesure pour des problèmes spécifiques. Les approches que nous allons prsenter s'attachent prcisment à éviter cette intervention du concepteur pour définir une heuristique ad hoc.

Parmi les planificateurs les plus récents, nous pouvons citer comme planificateurs à base d'heuristique ceux de la famille `HSP` [10, 9], `GRT` [64], `FF` [33, 30] et `AltAlt` [57, 58]. Dans tous ces cas l'heuristique développée vise à donner une indication de la distance dans l'espace d'états entre l'état considéré et un état objectif. Les grandes lignes de chaque planificateur sont décrites ci-après. La plupart des heuristiques calculables automatiquement correspondent à la résolution d'un problème relaxé.

2. Cette résolution peut être évitée en appliquant quelques itérations de la valeur sans changer la politique de manière à approximer la valeur de la politique courante. Cela représente un nombre controlable d'itérations de la valeur.

2.3.1 HSP

Pour construire son heuristique, `HSP` utilise une relaxation du problème de planification à traiter. L'idée est la suivante. Supposons que les actions n'aient aucun effet négatif. A partir d'un état donné, combien d'actions contiendrait le plus petit plan pour atteindre un état objectif?

Cette mesure est un minorant de la valeur dans le problème réel. Elle permet donc de guider la planification en orientant systématiquement la recherche vers les états semblant les plus proches d'un état objectif.

Malheureusement, la résolution du problème relaxé est elle-même trop complexe. En conséquence, les longueurs de plans y sont estimées plutôt que calculées précisément. Ceci est accompli en supposant les réalisations de tout sous-but indépendantes entre elles. Ainsi, au lieu de calculer la longueur minimale d'un plan menant d'un état s à une conjonction de buts $b_1 \wedge b_2 \wedge ... \wedge b_n$, sont calculées les longueurs minimales de plans menant respectivement à b_1, b_2... b_n. Ces valeurs sont ensuite combinées pour obtenir l'estimation finale utilisée dans l'heuristique.

Deux types de combination ont été retenues par H. Geffner et son équipe. Dans un cas, la valeur maximale des valeurs liées à chaque sous-but d'une conjonction est retenue pour la conjonction elle-même. L'heuristique est alors appelée *max heuristic* et notée H_{max}. H_{max} est admissible (cf. annexe A), car le coût pour atteindre la conjonction de buts ne peut être inférieure au coût nécessaire pour atteindre chacun d'entre eux individuellement.

Dans l'autre cas, la somme des valeurs individuelles est retenue. Cette heuristique est appelée *additive heuristic* et notée h_{add}. Cette heuristique n'est pas admissible. En effet, elle correspond a des plans dans lesquels chaque sous but est obtenu l'un après l'autre, indépendamment. Cependant, le fait de rendre vrai une proposition peut en rendre une autre vraie du même coup, ou une partie du travail réalisé pour l'une peut être réutilisé pour l'autre. En conséquence, des plans plus courts que ce qu'indique h_{add} peuvent exister.

L'estimation de l'heuristique est faite avec un algorithme de programmation dynamique - cf. section précédente (2.2)- (une adaptation aux propositions - au lieu des états - de l'algorithme de Bellman-Ford [2] qui est une forme particulière d'itération de la valeur, en avant).

Soit P un ensemble de propositions : $P = p_1 \wedge p_2 \wedge ... \wedge p_n$

Comme nous l'avons vu, le coût pour obtenir P à partir de l'état s se définit de deux manières :

$$c_s^{max}(P) = max_{p \in P} c_s(p)$$

$$ou$$

$$c_s^{+}(P) = \sum_{p \in P} c_s(p)$$

Soit O l'ensemble des objectifs, et h l'heuristique utilisée, pour tout état s, nous définissons $h(s) = c_s(O)$. C'est ce qu'il faut calculer avec l'algorithme suivant.

L'estimation d'atteindre la proposition p depuis l'état s se définit récursivement de la manière suivante :

$$Si\ p \in s\ alors\ c_s(p) = 0$$

$$sinon\ c_s(p) = min_{a \in A(p)}[1 + c_s(Prec(a))]$$

où $A(p)$ est l'ensemble des actions ayant p pour effet positif, et $Prec(a)$ est l'ensemble des préconditions de a.

L'itération de la valeur évalue la valeur de l'heuristique pour un seul état s et les objectifs du problème de planification. Elle se déroule comme suit.

Pour tout p et l'état s considéré, $c_s(p)$ est initialisé à 0 si $p \in s$ ou à ∞ sinon. Pour les mises à jour, à chaque fois qu'une action a peut être exécutée dans s, alors ses effets positifs sont ajoutés à s pour obtenir les états devant être considérés ensuite. C'est ainsi qu'est exploré un arbre d'états accessibles depuis l'état initial. La valeur de $c_s(p)$ est modifiée ainsi :

$$c_s(p) \leftarrow min[c_s(p), 1 + c_s(Prec(a))]$$

Bien sûr, le calcul s'arrête quand les valeurs de $c_s(p)$ se stabilisent.

La première version de `HSP` [8] utilise l'heuristique h_{add} pour guider un algorithme de recherche de type *hill climbing* (cf. annexe A). A chaque pas, $h(s)$ est calculée pour tout état s accessible depuis l'état initial. Pour tout état, l'action menant vers le meilleur état est retenue, et ainsi de suite. En cas d'ex aequo, le choix est fait aléatoirement. Ce planificateur a montré de très bonnes performances de rapidité. Cependant il n'est ni complet, ni optimal en termes d'optimisation du nombre de coups.

Au contraire, `HSP2` [9] est complet. Il utilise lui aussi l'heuristique h_{add}, mais pour guider une recherche de type meilleur d'abord. Ce faisant, ses performances se sont même améliorées par rapport à `HSP`. L'évaluation de chaque feuille sur laquelle se base le choix de la branche à développer, se fait avec la fonction suivante :

$$f(n) = g(n) + W h_{add}(n) \tag{2.3}$$

$g(n)$ représente le coût accumulé. W pondère l'influence de l'heuristique sur l'acquis.

Notant que le calcul de l'heuristique à chaque pas prend plus de 80% du temps de planification d'`HSP` ou `HSP2`, Geffner et al. ont développé une nouvelle version - `HSPr`. Ce planificateur travaille en arrière (à partir des objectifs, vers l'état initial).

La distance de chaque proposition à l'état initial est calculée une fois pour toute comme précédemment (en calculant $c_{s_{init}}(p)$ pour toute proposition p). Au cours de la planification, pour tout état s,

$$h_{add}(s) = \sum_{p \in S} c_{s_{init}}(p)$$

.

En continuant à planifier en avant, il serait plus difficile de déterminer les distances pertinentes une fois pour toutes. En effet, il faudrait calculer pour chaque proposition p sa distance aux objectifs. Cependant, les objectifs ne décrivent pas un unique état. De ce fait, les objectifs pourraient paraître inatteignables depuis certaines propositions, alors que des états contenant ces objectifs sont en fait atteignables rapidement à partir des états contenant les propositions considérées.

Néanmoins, la planification en arrière introduit d'autres difficultés non présentes en avant. En particulier, des états incohérents peuvent être générés. L'heuristique h_{add} n'est pas assez performante pour les détecter en leurs attribuant une distance infinie. Pour pallier ce problème, certaines paires de propositions ne pouvant apparaître simultanément sont détectées. Tout nœud de la recherche en contenant une est systématiquement abandonné.

Pour le reste, `HSPr` se comporte comme `HSP2`, en utilisant une recherche de type meilleur d'abord. Son taux de génération de nœuds est plus élevé (puisqu'il n'a pas à recalculer l'heuristique à chaque fois). Ainsi, sur plusieurs domaines, il montre de meilleures performances qu'`HSP2`. Cependant sur certains autres il génère beaucoup plus de nœuds et `HSP2` reprend le dessus en termes de performances.

2.3.2 GRT

L'heuristique utilisée par *GRT* (cf. [64] - largement repris dans [65], ou [66]), h_{GRT}, estime en arrière (à partir des objectifs) la distance de chaque proposition aux objectifs. Sa principale originalité est de prendre en compte en partie les interactions entre propositions quand il est recherché de les produire en conjonction.

Pour ce faire les auteurs introduisent la notions de *faits en relation* (*related facts*). Un fait q est relié à un fait p si produire p cause aussi la production de q.

Lorsque la distance aux buts d'une conjonction de propositions est calculée, cette notion est prise en compte. Supposons d'abord que toutes les propositions considérées soient reliées les unes aux autres de telle manière qu'elles soient toutes nécessaires à la production d'une même proposition P. Nous raisonnons en chaînage arrière. Les effets sont donc les préconditions du chaînage avant. Il s'en suit que P est en fait une précondition inévitable à l'établissement du reste du groupe. Surtout, cela signifie qu'en rejoignant les objectifs depuis P les autres propositions seront réalisées. Seule la distance de P aux objectifs est donc retenue pour l'ensemble des propositions.

Si cette configuration particulière n'est pas celle du groupe de propositions considéré, alors celui-ci est divisé en sous-groupes reprenant cette configuration. Chaque sous-groupe se voit attribuer un coût de la manière définie précédemment. Ensuite, les coûts des différents sous-groupes sont sommés pour obtenir la distance finale.

Notons que le calcul de cette heuristique nécessite une adaptation des objectifs. En effet, les objectifs ne décrivent pas totalement un état précis. Considérons par exemple un problème concernant notre robot *Robby* et trois balles - $B1$, $B2$ et $B3$. L'objectif est d'avoir les balles $B1$ et $B2$ dans la salle $S2$. Ici, ni la position finale de *Robby*, ni celle de $B3$ ne sont stipulées. Lors de la régression à partir des objectifs, ceci peut rendre certaines propositions inatteignables, empêchant de calculer leurs distances aux objectifs (ici, c'est le cas des positions de $B3$).

Pour pallier ce problème, les auteurs joignent aux objectifs toutes les propositions concernant des aspects non précisés. L'ensemble de propositions obtenues ne décrit pas un état ni même un ensemble d'états valide. En effet, dans notre exemple, toutes les propositions concernant la position de *Robby* et de $B3$ seraient introduites - alors que le robot, comme la balle, ne peuvent être qu'en un endroit à la fois.

Cette heuristique a été testée avec un algorithme de recherche de type *meilleur d'abord* (cf. [64]).

2.3.3 FF et AltAlt

Dans le cas de *FF* et *AltAlt*, le calcul des heuristiques est très lié à la *planification disjonctive*. Ils seront donc présentés plus en détails avec celle-ci au chapitre 4. Toutefois, il est important de noter d'ores et déjà qu'ils utilisent ces heuristiques dans l'espace d'états, de manière non disjonctive.

L'algorithme de recherche utilisé par *FF* est une sorte de croisement entre une recherche de type *hill climbing* et une recherche en *largeur d'abord*. J. Hoffman la nomme *enforced hill climbing method*. A partir d'un état s une recherche de type *hill climbing* choisirait à l'aide de l'heuristique le meilleur successeur immédiat. Ici, s est la racine d'une recherche en largeur d'abord qui s'arrête quand un état s' a été trouvé tel que $h(s') < h(s)$. Ces recherches en largeur d'abord permettent de parcourir les "plateaux" de h. Par ailleurs, les états y sont mémorisés de sorte qu'aucun n'est développé en double.

Cette approche évite les plateaux et certains minima locaux. Par ailleurs elle est complète dans tous les problème sans *deadlock* - sans "voie sans issue" en somme. Ainsi, les seuls problèmes dans laquelle elle est incomplète sont ceux pour lesquels il existe un état atteignable depuis l'état initial à partir duquel le but ne peut plus être atteint. Si la recherche atteint cet état, elle ne pourra jamais se remettre sur la bonne voie. Il semble cependant que la plupart des problèmes traités dans la compétition de planification de AIPS'2000 ne présentaient pas de *deadlock* ([31]).

AltAlt est une approche hybride entre la planification heuristique (au travers d'*HSPr*) et la planification disjonctive. Sa description fait trop appel à la planification disjonctive pour pouvoir s'y engager dès à présent. Nous nous bornerons donc à donner la signification du nom *AltAlt*, d'après les auteurs eux-mêmes ([58]) : *A Little of This and A Little of That* (*Un peu de ceci et un peu de cela*) - voilà qui annonce clairement une certaine hybridation!

2.4 Extensions à l'incertain et à l'optimisation

En premier lieu, notons que l'espace d'états est particulièrement adapté aux représentations de valeurs à optimiser ou de l'incertitude. En effet, la fonction d'utilité, par exemple, est une fonction des états vers les nombres réels. La représentation de l'incertitude peut être un peu plus complexe puisqu'il faut prendre en compte les différentes alternatives, éventuellement accompagnées de distributions de probabilités, mais ce sont toujours des états qui sont manipulés.

Cependant, rappelons-nous que ces problèmes sont plus durs que le problème classique. En particulier, l'optimisation (et donc le sous-ensemble des problèmes incertains qui s'y ramènent) ne peut pas s'accomoder de l'obtention d'une solution et nécessite donc une recherche complète de l'espace pour garantir que l'optimum a été trouvé. Les difficultés déjà rencontrées pour le problème classique liées à la taille de l'espace d'états vont donc être renforcées dans ce contexte.

Les méthodes arborescentes peuvent s'adapter en utilisant les algorithmes de recherche de type *meilleur d'abord* ou A^* (cf. annexe A). Remarquons d'ores et déjà que l'algorithme de *séparation-élimination Branch and Bound* en profondeur d'abord (cf. annexe A) ne peut pas être appliqué en planification (à moins de borner la profondeur de recherche). En effet, comme nous l'avons vu, les plans infinis empêchent d'utiliser des stratégies en profondeur d'abord. Pour résoudre un problème de satisfaction de contraintes, par exemple, ce problème ne se pose pas car le nombre de variables à instancier est borné (cette borne constitue la profondeur

maximale de l'arbre). Il faut donc l'adapter en *profondeur itérée*, ou encore l'initier avec une première valeur ad'hoc pour la borne. Placer cette valeur initiale trop bas, peut amener le planificateur à ne trouver aucun plan. La placer trop haut, peut laisser le planificateur s'engouffrer trop profondément dans de mauvaises branches.

La programmation dynamique est par essence adaptée à l'optimisation. En fait, elle est conçue pour cela. C'est le problème classique de planification qui y est traité comme un cas particulier d'optimisation. Par rapport à notre présentation, seul le critère doit être modifié. En plus de la récompense terminale, des coûts ou des gains peuvent être attribués aux transitions. Les équations d'optimalité (équation 2.1) se modifient pour cumuler ces valeurs, ou bien les minimiser ou les maximiser, le long du chemin.

La programmation dynamique a été étendue au traitement de l'incertitude. Les problèmes totalement observables peuvent être traités par les *Processus Décisionnels de Markov* ([62]) (*PDM* ; en anglais *MDP - Markov Decision Process*). Les problèmes seulement partiellement observables sont eux traités par les *Processus Décisionnels de Markov Partiellement Observables* ([12])(*PDMPO* ; en anglais *POMDP - Partially Observable Markov Decision Process*). Dans ce dernier cas, l'itération de la valeur ou de la politique ne se fait plus sur l'espace d'états à proprement parler, mais sur l'espace de croyances sur les états.

Les approches heuristiques sont elles aussi faites pour l'optimisation. Il suffit d'adapter la définition de l'heuristique. Toutefois, les méthodes de type *Hill Climbing* restent incomplètes. De plus dans ce contexte, elles ne garantissent pas l'optimalité des solutions trouvées.

2.5 Conclusion

Rechercher un plan dans l'espace d'états est une approche très générale de la planification. Elle permet d'appréhender le problème classique de planification, l'optimisation, et le traitement de l'incertitude.

En contre-partie, elle peut générer des calculs inutiles. C'est par exemple la multiplication de recherches identiques dans les algorithmes *FSS* ou *BSS*. Dans le cadre de la programmation dynamique, c'est le fait de résoudre un problème plus important que celui dont la solution est effectivement attendue. En programmation dynamique arrière, par exemple, le problème de rejoindre l'état *but* à partir de l'état *init* est entraîne nécessairement la résolution du problème - plus large - de rejoindre l'état *but* à partir de tout état possible du système.

Cependant, certaines lourdeurs de calcul proviennent simplement d'exigences de complétude. C'est particulièrement aigu dans le cadre de l'optimisation. Les performances dans le cadre de telles tâches ne peuvent être comparées à celles obtenues pour des problèmes de satisfaction.

Néanmoins, même pour les problèmes les plus difficiles, il est possible d'améliorer les performances. L'idée principale est de ne pas explorer avec la même minutie tout l'espace lorsqu'il est possible de savoir a priori que certaines régions ne contiennent pas la solution. Les approches heuristiques permettent précisément d'exploiter de telles évaluations a priori. L'enjeu de ce type d'approches est d'éviter le plus d'exploration inutile, tout en garantissant qu'aucune région n'est écartée à tort.

Chapitre 3

Planification dans l'espace des plans

Nous avons vu que la planification peut se faire dans l'espace des états. Les approches paraissent plus ou moins élaborées - depuis ce qui peut être vu comme une simulation (*FSS*) jusqu'à des méthodes d'énumération implicite (programmation dynamique).

Une autre voie existe aussi. Il s'agit de la planification dans l'espace des plans. Cette fois les états ne sont pas considérés pour eux-mêmes. Le travail s'effectue en manipulant directement des plans et en considérant leurs possibles modifications. D'une certaine manière, ce type d'approche peut être vu comme un pur fruit des recherches en planification. Ces approches se basent sur des réflexions sur ce qu'est un plan, et sur les oprations qui peuvent être appliquées à un plan.

Ce chapitre présente d'abord les notions essentielles à la définition et à la manipulation des plans. Celles-ci nous permettent de présenter les mécanismes de la planification dans l'espace des plans. Enfin, nous aborderons les planificateurs dans l'espace des plans s'attachant aux extensions du problème classique de planification - l'optimisation et la prise en compte de l'incertitude.

3.1 Définitions et notations

Le plan classique a été défini dans le premier chapitre (cf. 1.1). Les travaux sur la planification dans l'espace des plans se sont focalisés sur cette définition pour en tirer les principes spécifiques à la génération et modification d'un plan.

Notation : Soit (A,O,I,B) un plan.
A est l'ensemble d'actions présentes dans le plan ; O l'ensemble de contraintes d'ordonnancement sur les actions de A ; I les caractéristiques, ou propriétéd, de l'état qui doivent être vérifiées pour que le plan soit exécutable ; B les caractéristiques qui sont vérifiées à l'issue de l'exécution du plan.

Il existe deux types de contraintes d'ordonnancement.

Définition : ***contrainte de précédence***
Dire que l'action A_1 précède A_2 signifie que l'action A_1 doit être exécutée avant A_2, mais pas nécessairement juste avant.

Définition : *contrainte de contigüité*
Si l'action A_1 précède A_2 et que A_1 doit être exécutée juste avant A_2 - sans autre action entre les deux - alors A_1 et A_2 sont *contigües*.

Les contraintes d'ordonnancement peuvent créer ou éviter des *conflits* entre actions.

Définition : *conflit entre actions*
Deux actions sont dites *en conflit* si et seulement si, compte tenu des contraintes d'ordonnancement les concernant, il est possible que l'une détruise une précondition de l'autre.

La planification dans l'espace des plans nécessite une représentation en intension. Nous parlerons de propositions pour désigner les caractéristiques ou propriétés des états.

Définition : *proposition soutenue vs. proposition ouverte*
Considérant un plan (A,O,I,B) donné, une proposition P est dite *soutenue* si l'une des deux conditions suivantes est remplie :

1. $P \in I$;
2. P est l'effet d'une action de A.

Dans le cas contraire, il est question de proposition *ouverte*.

Afin de décrire la planification dans l'espace des plans, plusieurs types de plans doivent être distingués.

Définition : plan solution
Un plan (A,O,I,B) est solution d'un problème P si les 5 conditions suivantes sont remplies :

1. les caractériques listées dans I sont valides dans l'état initial de P ;
2. B contient les objectifs de P ;
3. pour toute action dans A, aucune de ses préconditions n'est ouverte ;
4. considérant A et O, il n'existe aucun conflit.
5. toutes les propositions de B sont soutenues.

Définition : plan partiel
Considérant un problème P, tout plan remplissant les deux premières conditions pour être plan solution mais qui ne remplit pas l'une au moins des trois autres est un *plan partiel*.

Définition : plan vide
Un plan (A,O,I,B) est dit *vide* si A et O sont vides.

Définition : préfixe de plan

Un *préfixe de plan* (A,O,I,B) est un plan partiel dans lequel :

1. les contraintes de O constituent un ordre total sur A ;
2. toute action de A est contigüe à une autre action de A ;
3. les préconditions de la première action appartiennent à I.

Définition : suffixe de plan

Un *sufixe de plan* (A,O,I,B) est un plan partiel dans lequel :

1. les contraintes de O constituent un ordre total sur A ;
2. toute action de A est contigüe à une autre action de A ;
3. tout ou partie des effets de la dernière action appartiennent à B.

3.2 Parcours de l'espace des plans : mouvements élémentaires

Un processus de planification dans l'espace des plans commence par le plan vide (cf. fig. 3.1).

FIG. 3.1 – *La planification dans l'espace des plans débute généralement à partir du plan vide. Ce plan ne contient aucune action. Les propositions soutenues (en caractères droits sur le schéma) appartiennent toutes à l'état initial (ici à gauche) :* $\{i_1,i_2,...,i_n\}$. *Les propositions à établir (en caractères italiques sur le schéma) constituent les objectifs (ici à droite) :* $\{b_1,b_2\}$

Le parcours de l'espace des plans se fait de proche en proche par deux types d'interventions :

1. ajout d'actions ;
2. ajout de contraintes d'ordre sur les actions déjà introduites.

3.2.1 Ajout d'une action au plan partiel courant

Lorsqu'une action est introduite dans le plan partiel courant, ses effets sont ajoutés à l'ensemble des propositions soutenues. Si certaines parmi elles étaient ouvertes, alors elles sont retirées de cette liste. Les préconditions de l'action introduite, en revanche, sont ajoutées aux propositions ouvertes. Si certaines d'entre-elles sont déjà soutenues, il peut être choisi d'utiliser les instances soutenues (cf. fig. 3.2).

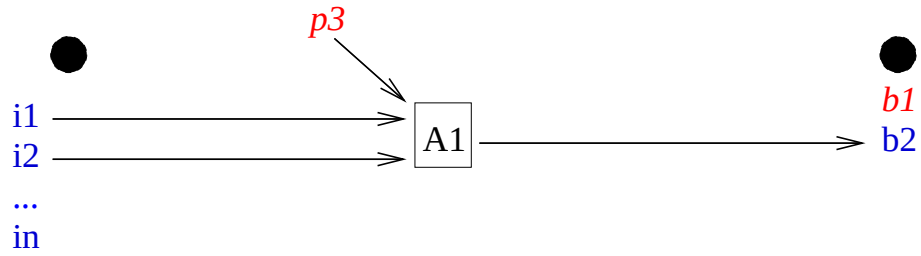


FIG. 3.2 – *Afin de supporter une des propositions à établir, une action a été introduite. Il s'agit ici de l'action A_1. Elle établit la proposition b_2. Elle a trois préconditions : i_1, i_2, et p_3. Les deux premières sont déjà établies, mais la dernière ne l'est pas - elle est donc ajoutée à l'ensemble des propositions à établir. En revanche, b_2 peut désormais être retirée de cet ensemble. Au final, nous obtenons l'ensemble de propositions établies $\{i_1,i_2,...,i_n,b_2\}$, et l'ensembles des propositions à établir $\{b_1,p_3\}$*

3.2.2 Ajout d'une contrainte d'ordre au plan partiel courant

L'ajout d'une contrainte d'ordonnancement ne change rien aux listes des propositions soutenues et ouvertes. En revanche, les contraintes de précédence spécifiquement linéarisent un peu plus le plan (puisque les deux actions concernées ne peuvent plus être exécutées en parallèle).

L'intérêt le plus notable de l'introduction d'une contrainte d'ordre est de pouvoir résoudre un conflit entre actions. La résolution intervient en interdisant les séquencements qui posent problème.

Par exemple, lorsqu'une action $A1$ "consomme" une précondition d'une autre action $A2$, c'est-à-dire la requiert comme précondition et la détruit, une contrainte d'ordonnancement doit être introduite de sorte qu'il soit garanti que $A2$ précède $A1$ (cf. fig. 3.3).

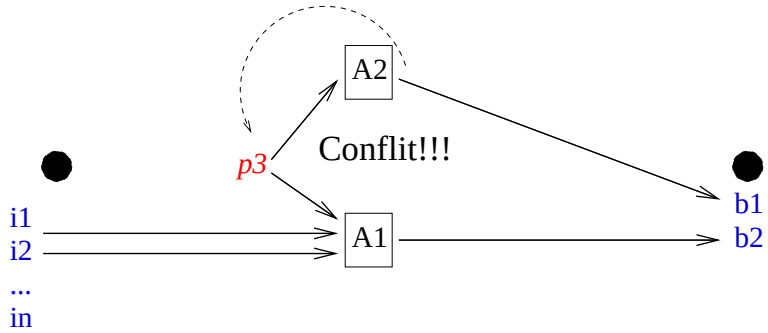


FIG. 3.3 – *Une deuxième action - A_2 a été introduite dans le plan. Elle permet d'obtenir le sous-but b_1, celui-ci est donc retiré des propositions à établir. Elle nécessite une seule précondition - p_3 (laquelle est déjà présente dans la liste des propositions à établir). Le lien en pointillés qui sort de A_2 pour revenir sur P_3 signifie que A_2 a besoin de p_3 sans la détruire. Malheureusement p_3 est détruite par l'exécution de l'action A_1. Dès lors, il y a conflit entre A_1 et A_2 car la première détruit une précondition de la seconde. Une solution à ce conflit sans introduire de nouvelles actions est d'imposer que A_1 s'exécute après A_2. Cette contrainte de succession est introduite dans le plan.*

3.3 Stratégies de parcours de l'espace des plans

Il est important de comprendre que le parcours mis en œuvre par le processus de recherche permet de passer de plan en plan jusqu'au plan solution. A chaque étape correspond un plan partiel.

Contrairement à certaines approches dans l'espace des états (*FSS* par exemple), le cheminement lui-même de la recherche **ne définit pas un plan**. Il **mène** à un plan.

En effet, *FSS* parcourait l'espace d'état en état depuis l'état initial jusqu'à un état objectif. C'était ce cheminement lui-même qui constituait le plan solution. Ici il n'en est rien. La recherche parcourt l'espace depuis le plan vide jusqu'à un plan solution. Une fois ce plan solution trouvé, le cheminement pour y parvenir peut être oublié (sauf pour d'éventuelles préoccupations de révision du plan [28]).

Impossibilité : parcours de l'espace des plans en chaînage arrière

La recherche d'un plan solution en chaînage arrière dans l'espace des plans est impossible. En effet, elle implique de partir d'un plan solution - justement - pour remonter jusqu'au plan vide. S'il est possible de partir d'un plan solution connu, c'est qu'il n'y a pas lieu de le rechercher !

En dehors du chaînage arrière, toutes les méthodes de recherche de l'annexe A pourraient être utilisées pour chercher une solution dans l'espace des plans : chaînage avant informé ou non informé, hill climbing, etc. En effet, il est important de distinguer l'espace de recherche - composé d'états dans un cas, de plans partiels dans un autre etc. - des méthodes de parcours de cet espace, qui elles sont génériques.

Dans la pratique, seul le chaînage arborescent avant est utilisé. C'est-à-dire que la recherche démarre toujours avec le plan vide et parcourt systématiquement l'espace jusqu'à atteindre un plan solution s'il en existe.

Les règles de production sont l'ajout d'actions ou de contraintes d'ordonnancement. Ainsi, les plans partiels considérés pendant la recherche contiennent de plus en plus d'actions et de contraintes.

Une alternative à ce chaînage avant serait par exemple de partir d'un plan partiel non vide et de générer d'autres plans partiels par ajout d'actions ou de contraintes, mais aussi par retraits d'actions ou de contraintes. Cette approche pourrait être utile par exemple pour exploiter des plan solutions de problèmes similaires, ou pour la révision de plans ([28]).

En chaînage avant, diverses stratégies sont cependant possibles. Elles définissent l'ordre dans lequel les actions et les contraintes d'ordonancement sont ajoutées aux plans partiels.

3.3.1 Allongement de préfixes ou de suffixes de plans

Dans le cas d'allongement de préfixes de plans, les actions sont introduites en fonction des propositions soutenues. La première action introduite a donc toutes ses préconditions dans l'état initial. De plus chaque action insérée est ordonnée après la dernière de celles déjà introduites, avec une contrainte de contiguité.

La figure 3.4 montre le début d'un arbre de recherche dans l'espace des plans où la stratégie de parcours consiste à allonger des préfixes de plans partiels. Mise en parallèle avec la figure 2.1 montrant l'arbre de recherche de *FSS* pour le même problème, cette figure fait clairement apparaître la dualité entre espace d'états et espace de plans.

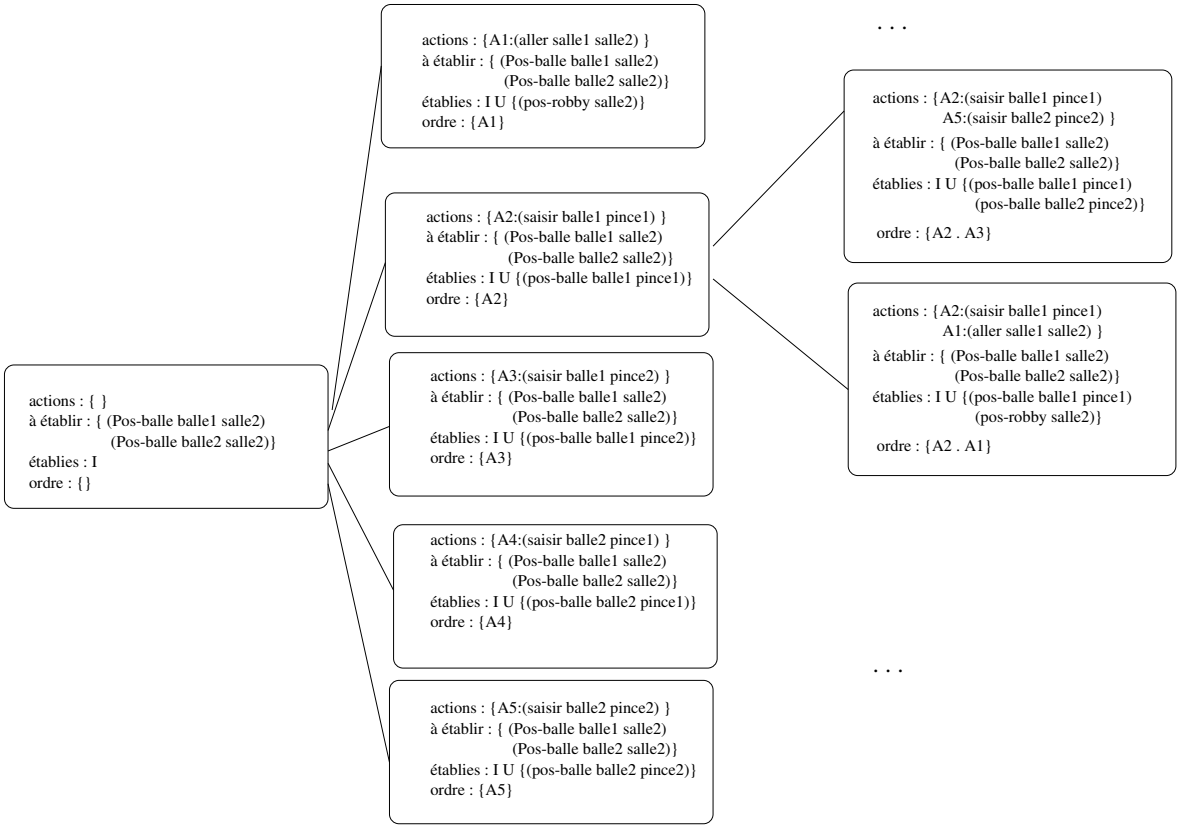


FIG. 3.4 – *Il s'agit ici d'un arbre de recherche par chaînage avant dans l'espace des plans. Seul le début de l'arbre est représenté. Le problème traité est le même que celui des figures 2.1 et 2.2. Ici le plan ne correspond pas à un chemin de l'arbre mais à une feuille. La stratégie de parcours de l'espace des plans mise en œuvre est de faire croître des préfixes de plans. Ainsi, toute action nouvellement sélectionnée est immédiatement placée juste après la dernière du plan partiel en cours.*

De la même manière, il peut être choisi d'allonger des suffixes de plans. Dans ce cas les actions sont introduites en fonction des propositions à soutenir. Chaque action insérée est ordonnée avant la dernière de celles déjà introduites, avec une contrainte de contiguité.

L'arbre de recherche pourrait alors utilement être mis en parallèle avec celui de *BSS*. Toutefois, la recherche dans l'espace des plans est bien toujours effectuée par chaînage avant. Il ne faut pas confondre la méthode de parcours de l'espace des plans et l'ordre des actions dans le plan.

Dans un cas comme dans l'autre, les plans générés sont basés sur des ensembles d'actions totalement ordonnés.

3.3.2 Stratégie de moindre engagement

Le principal avantage de la planification dans l'espace des plans sur les approches que nous avons vues dans l'espace des états est de pouvoir appliquer une stratégie de moindre engagement (*least commitment* en Anglais) [72] quant à l'ordonnancement des actions. Cet atout a conduit au développement des planificateurs à ordre partiel - souvent abrégés *POP* pour *Partial Order Planning* [3]. Parmi ces planificateurs, citons `SNLP` [50] et `UCPOP` [59].

Dans ce cadre, la stratégie d'insertion des actions dans le plan est guidée par les propositions à soutenir. Chaque action est introduite dans la liste des actions du plan sans lui assigner une position particulière.

C'est seulement dans le cas où elle entre en conflit avec d'autres actions que des contraintes d'ordre sont ajoutées. C'est là le cœur du *moindre engagement* : tant que cela n'est pas absolument nécessaire, aucune décision n'est prise quant à la position de l'action dans le plan. Il s'en suit que les planificateurs de ce type manipulent des plans où les actions sont seulement partiellement ordonnées.

La figure 3.5 représente partiellement un arbre de recherche construit par ce type d'approches pour le même problème que celui développé lors de la présentation des approches *FSS* et *BSS*.

Le principal intérêt du moindre engagement est de permettre de considérer plusieurs plans partiels en même temps. En effet, chaque position potentielle de l'action nouvellement introduite correspond à autant de plans partiels différents. En fait, le plan partiel manipulé est une représentation disjonctive de tous les plans partiels correspondant à ses divers ordonnancement totaux possibles.

Par ailleurs, le fait de ne pas imposer la contigüité des actions introduites découple le parcours de la recherche dans l'espace du parcours de l'agent représenté par le plan partiel courant lui-même.

Ce comportement est plus intuitif. En effet, considérons par exemple le cas d'un voyage à San Francisco pour assister à une conférence. A priori, parmi les première choses qui viennent à l'esprit, il y a le fait de devoir prendre un avion. Pour autant, le reste du plan jusqu'à l'embarquement n'est pas encore bien défini. En vrac, il va falloir réserver les billets, les payer, les retirer, préparer ses bagages, se rendre à l'aéroport - d'ailleurs quel aéroport? - etc.

De la même manière, un planificateur du type que nous considérons peut commencer par soutenir la localisation à San Francisco. Pour ce faire sera introduit un voyage en avion. Ensuite la proposition à soutenir peut être un autre objectif du problème ou une précondition de l'action introduite. Ainsi, deux pas consécutifs de planification peuvent concerner des étapes du plan qui sont éloignées temporellement. Certains points particulièrement difficiles

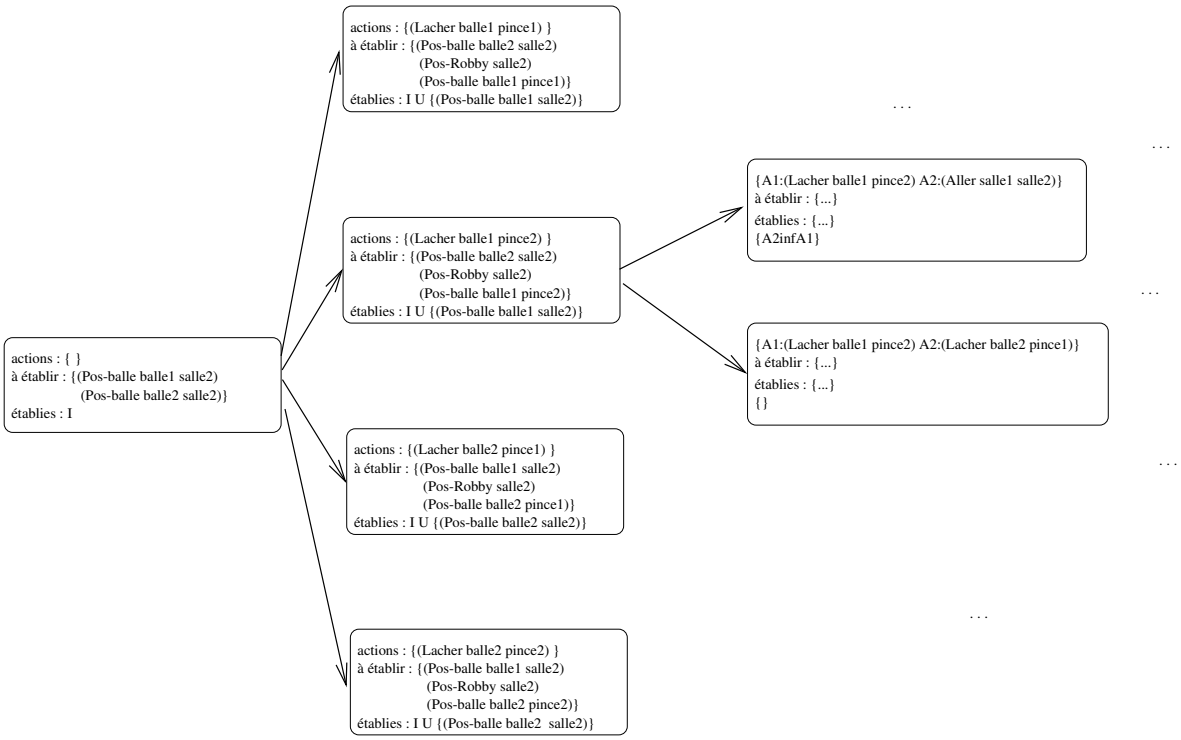


FIG. 3.5 – *Il s'agit ici d'un arbre de recherche par chaînage avant dans l'espace des plans. Seul le début de l'arbre est représenté. Le problème traité est le même que celui des figures 2.1 et 2.2. Ici le plan ne correspond pas à un chemin de l'arbre mais à une feuille.*

du problème traité peuvent ainsi être détectés dès le début de la planification, même s'ils concernent des étapes enfouies au milieu du plan.

Dans l'exemple de la figure 3.4, comme dans les approches de type *FSS*, ceci n'est pas possible. En effet, avant de pouvoir considérer le sous problème posé par l'étape n, il est nécessaire d'avoir construit un plan partiel jusqu'à l'étape $n-1$.

Les planificateurs dans l'espace des plans avec une stratégie de moindre engagement ont rapidement montré leur supériorité sur les approches de type *FSS* ou *BSS*. A partir de 1995, cependant ils ont été mis au second plan par la planification disjonctive d'abord (cf. chapitre 4), et les nouvelles approches heuristiques (cf. section 2.3 du chapitre 2).

3.4 Extensions à l'incertain et à l'optimisation

Le planificateur à ordre partiel prenant en compte l'incertitude le plus connu est `Buridan` [41] [42]. Il couvre à la fois une incertitude quant à l'état initial - observabilité partielle, et une incertitude quant aux effets des actions - actions non déterministes. Les plans recherchés doivent permettre d'atteindre les objectifs avec une probabilité supérieure à un certain seuil.

Son fonctionnement est celui d'un planificateur à ordre partiel tel que nous venons de le présenter. Cependant trois différences essentielles peuvent être notées. Elles concernent les propositions ouvertes, la résolution des conflits, et la validation d'un plan.

Jusqu'ici, une proposition ouverte devait être soutenue. A partir du moment où une action

introduite la comptait parmi ses effets (sans conflit), cette proposition était considérée soutenue. Dans le cadre probabiliste, les choses ne sont pas si simple. En effet, les liens de support entre actions et propositions sont associés à des probabilités, car les effets des actions sont incertains et dépendent en partie de l'état courant. Il s'en suit que les propositions sont "plus ou moins supportées". Ainsi, des actions de soutien peuvent être ajoutées à une proposition déjà soutenue dans le seul but d'augmenter sa probabilité d'occurence.

De la même manière, les conflits sont causés par des effets qui sont eux-même probabilistes. Ainsi, un conflit peut se résoudre en minimisant les chances d'occurence des effets en jeu. Les effets dépendant en partie de l'état courant, la technique de la *confrontation* consiste à forcer le support de propositions qui rendent le déclenchement des effets incriminés impossible.

Enfin, un plan n'est valide que s'il dépasse un certain seuil de probabilité de réussite. Il faut donc calculer la probabilité d'atteinte des objectifs. Plusieurs approches plus ou moins efficaces ont été développées pour remplir cette mission à partir d'un plan partiellement ordonné (les détails sont dans [41] et [42]).

Pour traiter la planification contingente, `Buridan` a été modifié pour donner `C-Buridan` [14]. Ce dernier reprend en grande partie l'approche d'un autre planificateur contingent : `CNLP` (pour *Conditional Non Linear Planning*) [60].

Contrairement, à `C-Buridan`, `CNLP` ne prend pas en compte des actions dont le comportement dépend de l'état dans lequel elles sont exécutées. L'incertitude quant aux effets n'est pas représentée numériquement mais au travers de trois niveaux représentés logiquement.

Par contre, tous deux utilisent la notion de *contexte* pour gérer une exécution contingente. Le contexte d'une action est une conjonction d'observations qui définit les circonstances dans lesquelles l'action considérée doit être exécutée. Au moment de l'exécution, les observations permettent de définir quel est le contexte compatible avec la réalité, et donc de choisir l'action appropriée.

3.5 Conclusion

La planification dans l'espace des plans se focalise sur le plan lui-même. En effet, un plan ici n'est pas le sous-produit de la manipulation d'états mais bien l'objet de toutes les attentions.

Ainsi, la distinction est bien faite entre le plan d'action tel qu'il sera exécuté par l'agent, et la stratégie de recherche de ce plan. La description et la manipulation des plans est elle-même affinée. En particulier, les actions d'une part et leur ordonnancement d'autre part sont clairement distingués.

Ce dernier point permet en outre de relâcher autant qu'elles peuvent l'être les contraintes d'ordonnancement des actions. Cette stratégie de moindre engagement quant au séquencement des actions permet de manipuler au travers de chaque plan partiel un ensemble de plans plus large. Cette possibilité se traduit par des gains sensibles de performance.

Dans ce sens, la stratégie de moindre engagement dans le cadre de la planification dans l'espace des plans est un premier mouvement vers la planification disjonctive.

Chapitre 4

Planification disjonctive

En 1995, le planificateur `Graphplan` a été présenté [5, 6], et a surpassé en performances toutes les approches existantes. Rompant avec ce qui se faisait en planification d'actions, il a ouvert la voie de la *planification disjonctive*. Depuis lors, tous les planificateurs de cette voie sont des descendants directs de `Graphplan`, optimisant son algorithme, améliorant ses représentations internes, ou ajoutant de nouvelles capacités à une machinerie "à la Graphplan". D. Weld a d'ailleurs écrit une synthèse de ce que l'on pourrait appeler les "années après Graphplan" [73].

Ce chapitre est important car la planification disjonctive occupe une place de premier rang dans la planification classique récente. Il est aussi important dans le cadre de ce document car nos travaux s'insèrent précisément dans le domaine de la planification disjonctive.

Nous allons présenter en premier lieu le planificateur `Graphplan`. Ensuite nous nous intéresserons aux sources de l'efficacité de la planification disjonctive. A cette fin, deux points de vues seront présentés : celui de la planification heuristique principalement soutenu par Geffner et al., et celui de la planification par raffinement principalement représenté par Kambhampati et al. . Enfin, nous présenterons les extensions de `Graphplan` pour la prise en compte de l'incertitude et de l'optimisation.

4.1 Graphplan

`Graphplan` utilise une représentation en intension du domaine de planification. Il l'exploite en construisant dans un premier temps un graphe explicite - le *graphe de planification*. La recherche d'un plan solution se fait dans un second temps, à l'intérieur du graphe de planification.

4.1.1 Eléments du graphe de planification

`Graphplan` construit un *graphe de planification.*

Définition : *Graphe de planification*

Un *graphe de planification* est un graphe explicite dans lequel peut s'effectuer la recherche d'un plan solution à un problème donné. Une fois construit, ce graphe doit suffire à une méthode de recherche adaptée pour trouver un plan solution s'il en existe un. L'objectif d'un graphe de planification est triple :

1. encoder toute l'information nécessaire à la résolution d'un problème ;
2. adapter la représentation de l'information pour la recherche d'une solution ;
3. éliminer autant que possible l'information inutile.

Le graphe de `Graphplan` est composé de deux catégories de nœuds :

1. des nœuds associés à des propositions (chacun de ces nœuds étant associé à une seule proposition du domaine de planification) ;
2. des nœuds associés à des actions (chacun de ces nœuds étant associé à une seule action du domaine de planification).

C'est un graphe bipartite entre ces deux ensembles de nœuds. Par ailleurs, c'est un graphe organisé en niveaux. Ainsi, alternent les niveaux d'actions et ceux de propositions (cf. figure 4.1).

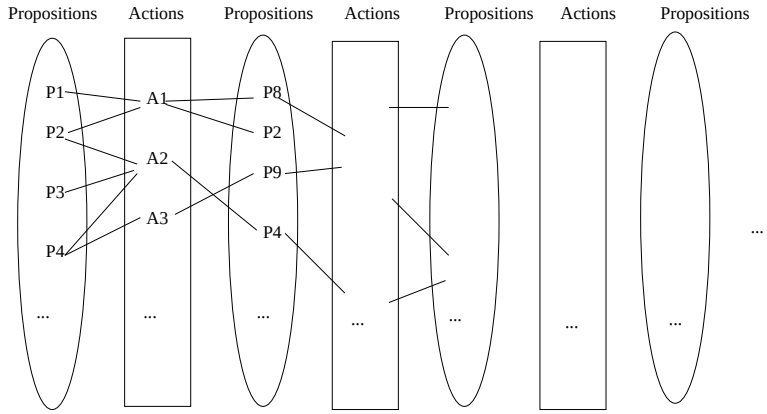


FIG. 4.1 – *Voici un graphe bipartite du type de celui construit par* `Graphplan`*. Les niveaux de propositions alternent avec les niveaux d'actions. Les nœuds d'un niveau sont reliés aux nœuds du niveau suivant. Au sein d'un même niveau, les nœuds ne sont pas reliés entre eux.*

La spécificité de graphe de `Graphplan` réside dans le choix des actions et propositions introduites dans les différents niveaux et dans la manière de connecter les nœuds.

`Graphplan` exploite au maximum les propriétés intensionnelles des représentations de type STRIPS. En effet, comme nous l'avons déjà évoqué, le fait qu'une action soit exécutable ou non ne dépend que de la validité des propositions de sa liste de préconditions. Peu importe l'état précis dans lequel elles sont valides. Ainsi, `Graphplan` ne maintient pas d'information explicite à propos des états eux-mêmes. Seules les propositions elles-mêmes sont considérées.

Chaque niveau de propositions (respectivement d'actions), ne contient au plus qu'une seule instance de chaque proposition (respectivement action).

Dans un niveau d'actions, chaque nœud est relié à des nœuds du niveau précédent, et à des nœuds du niveau suivant. Les premiers sont associés aux propositions préconditions de l'action considérée. Les autres sont associés à ses effets.

Définition : *Graphe en niveaux*
Soient A, B, et C, trois nœuds d'un graphe organisé en niveaux. Si A est relié à B, et B est relié à C, alors A ne peut pas être relié à C. A, B, et C appartiennent à des niveaux différents. Des nœuds d'un même niveau ne sont pas reliés entre eux.

Des actions supplémentaires à celles du domaine de planification peuvent être introduites dans le graphe. Il s'agit des actions `noOp`.

Définition : *Action NoOp*
Il y a autant d'actions *No-Op* différentes qu'il y a de propositions dans le domaine de planification considéré. Chacune d'entre elles a une seule proposition pour précondition, et la même pour effet.

Ces actions un peu particulières préservent les propositions qui ne sont pas modifiées par les autres actions. Elles ont aussi un rôle pour la partie ordonnancement de la tâche de planification. Nous y revenons en section 4.1.3.

4.1.2 Construction du graphe de planification

Le premier niveau est propositionnel. Il contient toutes les propositions de l'état initial. Pour toute action A du domaine et de l'ensemble d'actions *NoOp*, si ses préconditions sont présentes dans le niveau initial, alors A est introduite dans le deuxième niveau. Les effets de A sont alors introduits dans le troisième niveau. La figure 4.2 montre un exemple d'introduction d'une première action dans le graphe.

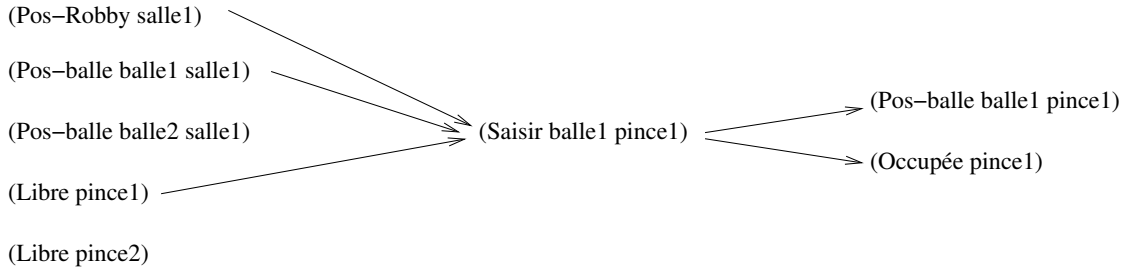


FIG. 4.2 – *Voici le début d'un graphe de planification de* **`Graphplan`**. *Le premier niveau est composé des propositions de l'état initial. Une première action est introduite et ses deux effets positifs au niveau propositionnel suivant. Les préconditions sont dans le premier niveau. L'action est dans le deuxième niveau. Les effets sont dans le troisième niveau. Ces nœuds sont reliés.*

Lors de l'introduction d'un effet P dans le troisième niveau deux comportements sont possibles (la figure 4.3 présente un exemple) :

1. Si P a déjà un nœud dans ce niveau alors aucun nouveau nœud n'est créé. Un lien est simplement introduit entre l'action considérée et le nœud déjà existant ;
2. Si P est totalement absent de ce niveau, alors un nouveau nœud est créé et relié à l'action considérée.

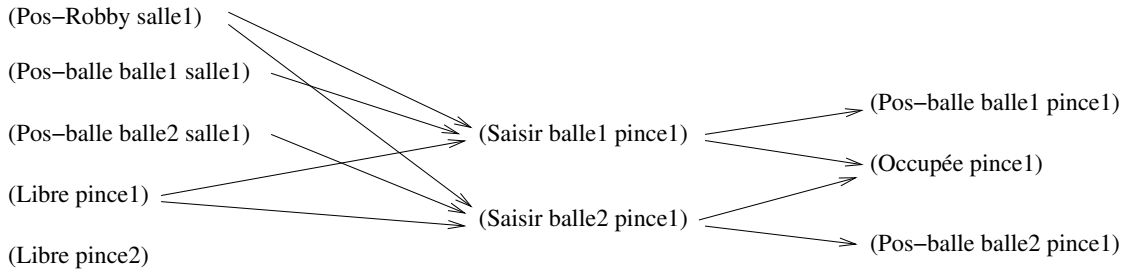


FIG. 4.3 – *Le graphe initié en figure 4.2 est ici développé un peu plus. Une nouvelle action a été introduite :* $(SaisirBalle2Pince1)$. *L'un de ses effets existe déjà dans le deuxième niveau propositionnel (*$(Occupee\ pince1)$*). Dans ce cas un simple lien est ajouté. L'autre effet n'a encore jamais été évoqué (*$(Pos - balle\ balle2\ pince1)$*). Un nouveau nœud est alors ajouté dans le deuxième niveau propositionnel.*

Une fois que toutes les actions exécutables dans l'état initial ont été introduites, la même procédure est à nouveau appliquée à partir du troisième niveau. La construction du graphe continue de niveau en niveau.

4.1.3 Encodage de l'espace de recherche et graphe de planification

Le graphe de planification encode de manière compacte
- d'une part, l'atteignabilité des propositions à partir de celles présentes dans l'état initial ;
- d'autre part, l'atteignabilité des possibilités de déclenchement des actions.

Sémantique : *Niveau propositionnel*
Les propositions présentes dans le n-ième niveau propositionnel forment une approximation de l'ensemble des propositions qui sont effectivement atteignable individuellement en $n-1$ coups. Plus précisément, une proposition P qui apparaît pour la première fois au n-ième niveau propositionnel ne peut pas être atteinte en moins de $n-1$ coups.

Sémantique : *Niveau d'actions*
Les actions présentes dans le n-ième niveau d'actions forment une approximation de l'ensemble des actions pour chacune desquelles il existe un plan dans lequel elle est exécutée au n-ième coup. Plus précisément, une action A qui apparaît pour la première fois au n-ième niveau d'actions ne peut pas être exécutée avant le n-ième coup.

Cette étude d'atteignabilité n'est qu'approchée parce que les propositions sont considérées pour elles-mêmes, indépendemment des états dans lesquels elles apparaissent. Chaque proposition correspond à un ensemble d'états. Un niveau propositionnel représente donc un

ensemble d'ensembles d'états. Par intersections, il est possible d'isoler les états valides auxquels appartiennent les propositions correspondant au niveau considéré. Cependant il existe aussi des intersections vides. Elles signifient que les propositions en jeu ne peuvent pas être valides en même temps à ce niveau-là. Il s'ensuit qu'une action les prenant pour préconditions ne pourrait pas être exécutée à cette étape-là. Or, les préconditions étant rassemblées dans le même niveau, l'action invalide va mécaniquement être introduite dans le niveau suivant, introduisant des effets peut-être impossibles même pris individuellement. Le graphe indique qu'ils sont accessibles au niveau n alors qu'ils ne le seront que plus tard (cf. figure 4.4).

Pour améliorer cette approximation, `Graphplan` calcule et mémorise pendant la construction du graphe de planification des relations binaires de mutuelle exclusion. Elles sont nommées *relations mutex.*

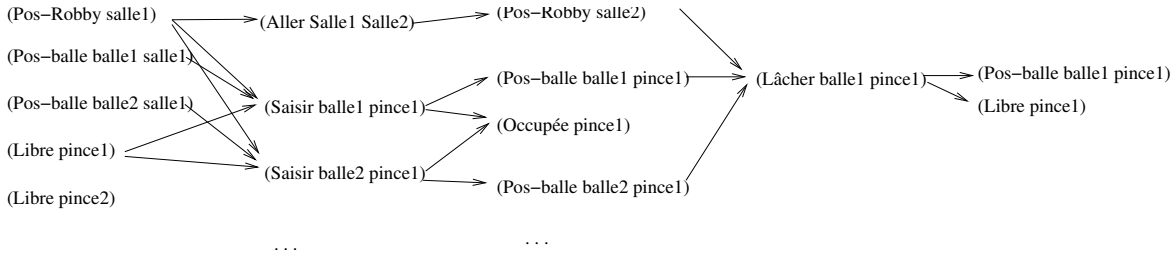


FIG. 4.4 – *Ici, le graphe de la figure 4.3 a été complété un peu plus. Attention : les premiers niveaux ne sont pas complets, il n'y a pas toutes les actions, ni les* NoOps. *Toutefois, il y a assez d'information pour constater que l'action* (*Lâcher balle*1 *pince*1) *est introduite au deuxième niveau d'actions alors qu'elle ne peut y être exécutée. En effet, cela suppose qu'à l'étape d'avant, Robby ait à la fois changé de salle et pris une balle dans la salle initiale.*

Définition : *Relations binaires de mutuelle exclusion (dites "relations mutex")*

Trois règles définissent les relations binaires de mutuelle exclusion entre actions et entre propositions au sein du graphe de planification de `Graphplan` :

1. *Interférence* : Deux actions sont mutuellement exclusives si l'une détruit une pré condition ou un effet positif de l'autre.
2. *Compétition de ressources* (*competing needs*) : Deux actions sont mutuellement exclusives si une précondition de l'une et une précondition de l'autre sont elles-mêmes marquées mutuellement exclusives au niveau précédent.
3. Deux propositions p et q sont mutuellement exclusives si toutes les actions soutenant p sont mutuellement exclusives avec toutes les actions soutenant q.

Toute action dont deux des préconditions sont mutuellement exclusives n'est pas introduite dans le graphe de planification. Avec ce système, l'action (*Lâcher balle*1 *pince*1 *salle*2) du deuxième niveau d'actions de la figure 4.4 n'est pas introduite. En effet, par exemple, les ses prconditions (*PosRobby salle*2) et (*PosBalle balle*1) sont mutuellement exclusives. Cette action apparaîtra dans les niveaux suivants.

Bien sûr, pour rendre l'étude d'atteignabilité parfaitement exacte, il faudrait aussi prendre en compte tous les ensembles mutuellement exclusifs - incluant ceux à trois éléments, quatre, et ainsi de suite. Pour s'en convaincre, la figure 4.5 exhibe un exemple dans lequel trois propositions sont mutuellement exclusives bien qu'elles ne le soient pas deux à deux. Le

résultat en est que $P4$ et $P5$ sont mutuellement exclusives sans que ce soit détecté. Cependant la prise en compte des seules relations binaires apporte déjà un gain important pour un coût de calcul raisonnable.

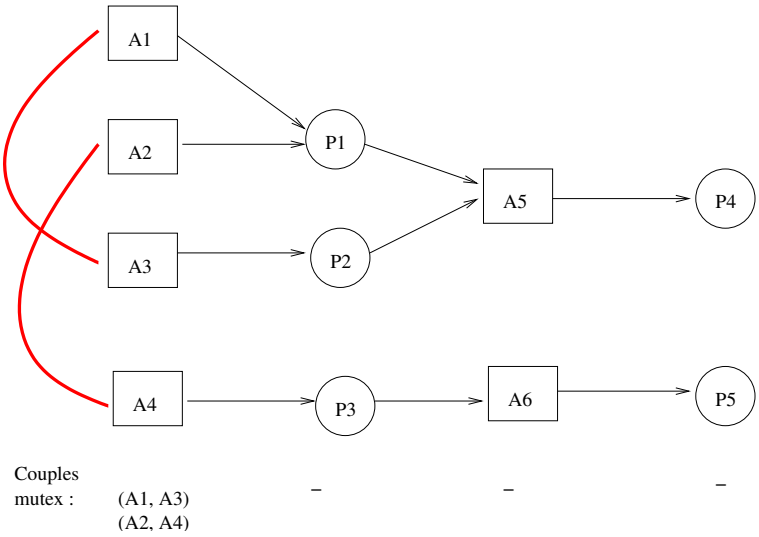


FIG. 4.5 – *Quatre niveaux successifs d'un graphe de planification sont présentés - soient deux itérations (actions et leurs effets, puis à nouveau actions et leurs effets). Le premier niveau compte quatre actions. Deux paires mutuellement exclusives y sont détectées. Parmi leurs effets, aucune relation binaire de mutuelle exclusion n'existe. En effet $P1$ et $P2$ ne sont pas mutuellement exclusives car $P1$ peut être supportée par $A2$ qui n'est pas mutuellement exclusive avec $A3$ - qui supporte $P2$. De la même manière, $P1$ et $P3$ ne sont pas mutuellement exclusives car $A1$ et $A4$ ne le sont pas. Enfin $P2$ et $P3$ ne sont pas mutuellement exclusives car $A3$ et $A4$ ne le sont pas non plus. Il n'empêche que $P1$, $P2$, et $P3$ ne peuvent pas être présentes en même temps. En effet, pour avoir à la fois $P1$ et $P2$, il faut que $P1$ soit soutenue par $A2$ et $P2$ par $A3$, seule combinaison compatible. Or la seule action pouvant soutenir $P3$ est $A4$ et $A4$ est mutuellement exclusive avec $A2$. En conséquence, les deux actions suivantes $A5$ et $A6$ sont mutuellement exclusives, mais* **Graphplan** *ne peut pas le détecter. Il en va de même pour les deux propositions finales $P4$ et $P5$.*

Les relations de mutuelle exclusion se répartissent en deux catégories [44, 53] :

1. les unes sont permanentes ;
2. les autres disparaissent au bout d'un certain nombre de niveaux.

Les relations permanentes de mutuelle exclusion reflètent des propriétés structurelles du domaine considéré. En particulier, elles encodent le fait que certaines ressources du domaine ne sont pas partageables, ou que les objets ne peuvent pas être en plusieurs endroits en même temps. En particulier, [44] souligne que deux actions mutuellement exclusives à cause de la règle *d'interférence* le sont de manière permanente.

La figure 4.6, montre un exemple où les relations de mutuelle exclusion encodent simplement le fait qu'un robot mobile ne peut pas être en deux endroits différents en même temps. Cela restera vrai tout au long de la planification!

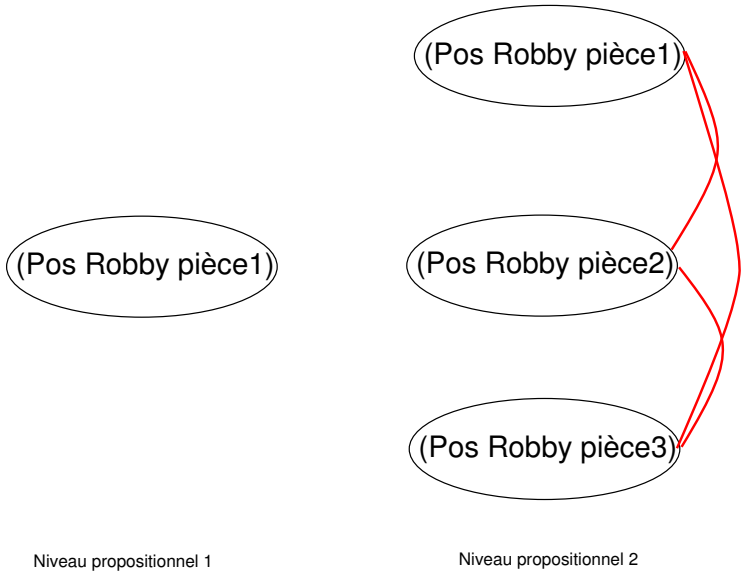


FIG. 4.6 – *A partir de la pièce 1, Robby peut y rester ou bien se rendre dans les pièces 2 ou 3 (le niveau d'actions a été omis pour ne pas alourdir le schéma). Bien sûr, dans la réalité, il ne peut se trouver que dans une pièce à la fois. Le domaine de planification a été écrit de telle sorte qu'il respecte cet aspect de la réalité. En conséquence, il apparaît des relations de mutuelle exclusion qui transcrivent cette restriction. Sur la figure elles sont représentées par les arcs entre les propositions. Ces trois relations de mutuelle exclusion ne disparaîtront jamais au cours de la construction du graphe - elles sont permanentes.*

Au contraire, certaines relations de mutuelle exclusion sont directement liées à l'avancement de la résolution du problème de planification considéré. Celles-ci vont disparaître au fur et à mesure que le processus de planification progresse. En effet, à mesure que l'étude d'atteignabilité s'étend à partir de l'état initial, des états jusqu'alors inatteignables sont rejoints. C'est précisément le but de la planification. Ces états nouvellement atteignables rendent atteignables des conjonctions de propositions qui ne l'étaient pas jusqu'alors. En conséquence, les relations de mutuelle exclusion qui décrivaient ces impossibilités jusqu'au niveau courant ne perdurent pas. La figure 4.7 présente un exemple de disparition d'une relation de mutuelle exclusion.

Contrairement aux relations de mutuelle exclusion, l'insertion des actions *NoOp* dans le graphe de planification est absolument nécessaire pour encoder correctement l'espace de recherche. Un premier rôle des actions *NoOp* est d'encoder ce qui ne change pas d'un niveau à l'autre. En Intelligence Artificielle, il s'agit d'un problème connu : *le problème du cadre*, ou *frame problem* en anglais. Lorsqu'une action est introduite, elle détruit des propositions et en crée d'autres. Elle laisse aussi inchangée toute une partie de l'état considéré.

Dans une approche de type *FSS*, les ajouts et suppressions de propositions sont appliquées à l'état considéré. L'état résultant est le nœud suivant dans l'arbre. Dans le graphe de planification, la structure d'état ayant disparue, il faut expliciter le maintien des propositions d'une étape à l'autre. C'est l'un des rôles des actions *NoOp*.

Le second rôle des actions *NoOp* est de permettre l'ordonnancement des actions lors de l'extraction du plan. En effet, à partir du moment où une action est introduite dans

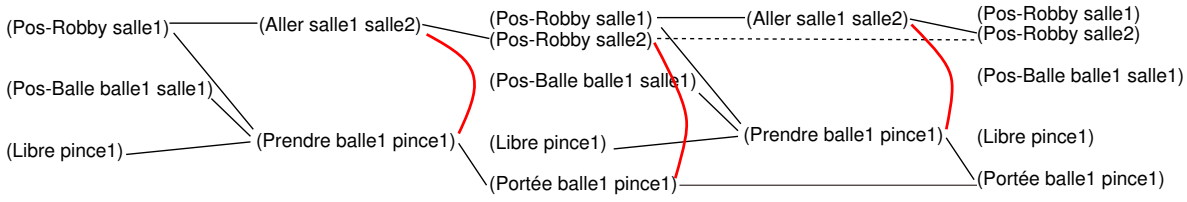


FIG. 4.7 – *Le début d'un graphe de planification est ici présenté. Seules les actions et propositions utiles pour notre propos ont été représentées. Au premier niveau d'actions,* $(Aller\, salle1\, salle2)$ *est mutuellement exclusive avec* $(Prendre\, balle1\, pince1)$*. Il s'en suit que* $(PosRobby\, salle2)$ *et* $(Portee\, balle1\, pince1)$ *sont mutuellement exclusives. En effet, Robby n'a pas pu changer de salle et prendre une balle en même temps. En revanche, au niveau suivant,* $(Portee\, balle1\, pince1)$ *est aussi soutenue par une action* NoOp*. Elle n'est donc plus mutuellement exclusive avec* $(PosRobby\, salle2)$*. En effet, Robby peut prendre la balle* ***puis*** *changer de salle.*

le graphe, elle le sera à chaque niveau ultérieur grâce aux actions *NoOp* qui maintiennent ses préconditions. Ce mécanisme rend sélectionnable l'action considérée à toutes les étapes ultérieures du plan. Ainsi, plutôt que de sélectionner une action sans se prononcer sur son emplacement exact, `Graphplan` sélectionne une action et en introduit une instance explicite à chaque niveau.

Ces mécanismes permettent de mener une étude d'atteignabilité sans introduire de découpage en états tel qu'il apparaît dans l'arbre de *FSS* par exemple. Ceci permet d'éviter les difficultés liées aux phénomènes de duplication de la recherche tels qu'ils apparaissaient dans les approches arborescentes.

En effet, toutes les propositions atteignables sont considérées pour elles-mêmes, et non plus au travers des états auxquels elles appartiennent. A ce titre, chaque proposition n'a au plus qu'une instance par niveau. De fait, un niveau propositionnel ne peut pas comprendre deux fois l'ensemble des préconditions d'une même action. Il s'en suit qu'aucune duplication n'est possible. En revanche, certains états inatteignables sont pris en compte inutilement.

Au final, la construction du graphe de `Graphplan` repose sur un compromis :

- par rapport à une recherche arborescente de type *FSS*, la construction de l'espace de recherche - le graphe ou l'arbre - est ici moins lourde ;
- en contrepartie l'étude d'atteignabilité encodée n'est plus exacte.

4.1.4 Exploitation du graphe de planification : recherche d'un plan

La phase de recherche arrière a pour but premier de vérifier si les objectifs sont effectivement atteignables. Si tel est le cas, elle extrait un plan.

Dans les approches présentées jusqu'à présent cette phase est triviale. En planification dans l'espace des plans, si une feuille satisfaisante est atteinte, elle contient le plan. Dans les approches de type *FSS* ou *BSS*, dès qu'un état objectif est atteint, un plan existe. Il s'agit simplement du chemin entre la racine et la feuille considérée. En programmation dynamique, une fois les valeurs stabilisées, le plan optimal s'extrait en suivant le meilleur potentiel depuis l'état initial jusqu'à un état objectif.

Ici, les avantages de la planification disjonctives en termes de coût de construction de l'espace de recherche se payent en terme de complexité de la recherche à y mener pour extraire un plan. Tout d'abord, du fait de l'approximation de l'étude d'atteignabilité, la présence des objectifs dans un niveau propositionnel ne garantit pas qu'ils soient effectivement atteignables.

Par ailleurs, chaque proposition est reliée à une *disjonction* d'actions de supports et une *disjonction* d'actions consommatrices. Ainsi, pour chaque étape du plan, il reste de nombreux choix à effectuer. Quelle action choisir pour supporter telle proposition? Quelle action pour telle autre? Evidemment, tous ces choix ne sont pas compatibles les uns avec les autres. Pour couronner le tout, ces incompatibilités ne sont pas toutes encodées au travers des relations de mutuelle exclusion mémorisées pendant la construction du graphe.

Recherche arrière : Condition de déclenchement

La recherche arrière est déclenchée à chaque fois qu'un nouveau niveau propositionnel contient tous les objectifs du problème considéré et que ceux-ci ne sont pas mutuellement exclusifs.

Dans la pratique, cela se traduit par une première phase de construction pure du graphe de planification jusqu'à atteindre un niveau contenant les objectifs. Cette phase est suivie d'une alternance de construction et de recherche arrière pour chaque nouveau niveau construit, jusqu'à ce qu'un plan soit trouvé ou qu'il soit prouvé qu'aucun plan n'existe.

La recherche procède comme suit. Pour chaque objectif à soutenir, il faut sélectionner une action la supportant telle que cette action n'est mutuellement exclusive avec aucune autre supportant les autres propositions du même niveau. Si un tel ensemble d'actions n'existe pas, la recherche s'arrête sur un échec. L'ensemble de propositions à soutenir est mémorisé comme impossible à soutenir à ce niveau.

Définition : *Ensemble Echec*

Au cours de la recherche de plan, un *ensemble-échec* est un ensemble de propositions qui a été prouvé comme n'étant pas effectivement atteignable à un niveau donné. Ces ensembles et leurs niveaux associés sont mémorisés pour éviter de reproduire la preuve d'échec lors d'une recherche ultérieure.

Dans le cas contraire, l'ensemble des préconditions des actions sélectionnées devient le nouvel ensemble de propositions à soutenir, cette fois au niveau précédent. Si un échec est retourné alors la recherche remonte d'un cran. En effet, peut-être qu'en choisissant d'autres actions pour soutenir les propositions considérées précédemment, un ensemble de préconditions pouvant être soutenues sera trouvé.

Pour bien comprendre cette phase de recherche, il faut voir qu'il y a en fait deux processus de recherche récursifs emboîtés l'un dans l'autre. Chacun est source de retours arrière (le terme *backtracks* est plus souvent utilisé).

1. Au sein d'un niveau, un premier processus choisit les actions pour soutenir les différentes propositions sélectionnées. Lorsqu'aucun choix valide n'est possible pour une proposition P, les choix déjà effectués pour les autres propositions sont remis en cause pour libérer d'éventuelles opportunités pour P.
2. L'autre processus, se déroule de niveau en niveau. Le choix des actions pour soutenir les objectifs définit un ensemble de propositions à soutenir au niveau précédent et ainsi de suite. Si l'un de ces ensembles de propositions n'a aucune solution, il faut remettre

en cause les choix effectués sur les niveaux déjà traités pour essayer de générer des ensembles différents dans l'espoir d'en trouver un avec une solution.

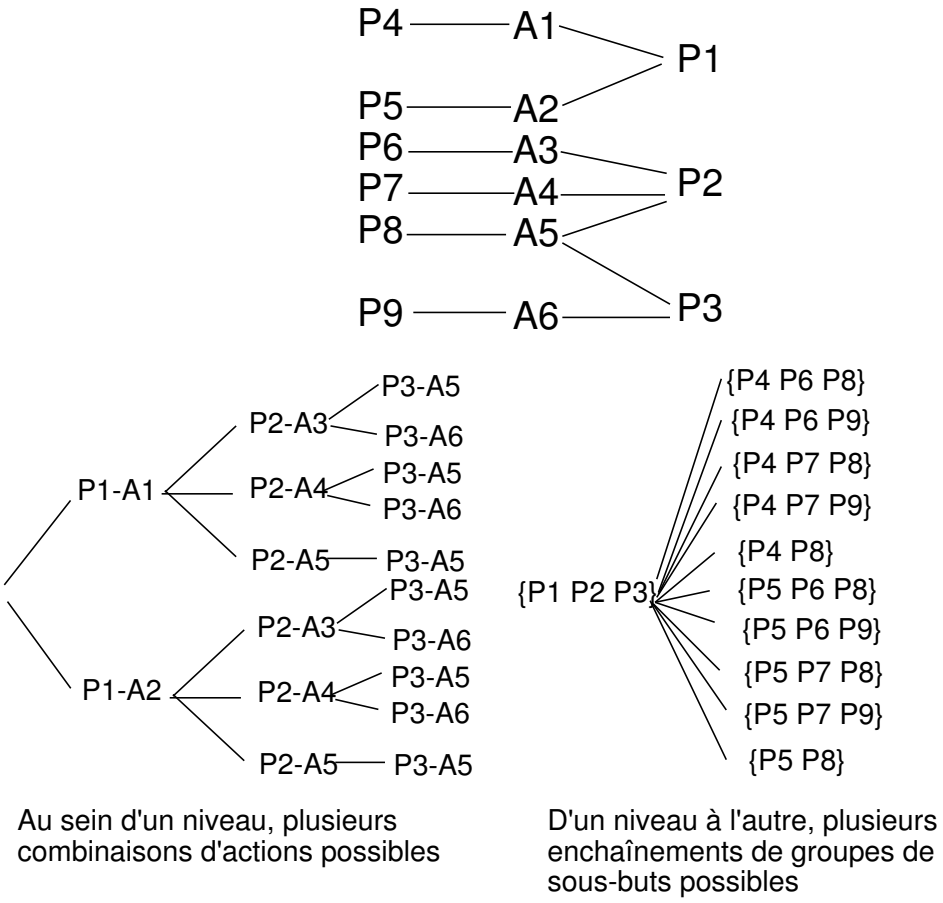


FIG. 4.8 – *Cet exemple met en relief les deux processus de recherche qui constituent la recherche arrière. L'arbre en bas à gauche détermine toutes les combinaisons d'actions possibles pour soutenir les trois objectifs $P1$, $P2$ et $P3$ du graphe de planification partiellement repésenté au dessus. Par exemple, la première combinaison (branche du haut) choisit $A1$ pour soutenir $P1$, $A3$ pour $P2$ et $A5$ pour $P3$. Les préconditions de cette combinaison d'actions sont $P4$, $P6$ et $P8$. Elles doivent être soutenues au niveau précédent pour que cette combinaision d'action soit retenue. L'arbre en bas à droite indique les ensembles de préconditions qui correspondent à chaque combinaison d'actions possible pour soutenir les trois objectifs.*

La figure 4.8 présente une illustration de ce double processus. Le schéma du haut correspond à une portion de graphe de planification. Les objectifs à établir sont $P1$, $P2$, et $P3$. Pour simplifier l'exemple, aucune relation de mutuelle exclusion n'est incluse ici.

D'une part, étant donné un ensemble de buts à soutenir au sein d'un niveau, il faut trouver une combinaison d'actions adéquates non mutuellement exclusives. L'arbre de choix est présenté en bas à gauche. En particulier, il est notable que choisir $A5$ pour $P2$ choisit du même coup $A5$ pour $P3$. En effet, $P2$ et $P3$ sont deux effets de $A5$ (voir la portion de graphe

de planification représentée). En conséquence, si $A5$ est introduite dans le plan pour soutenir $P2$ alors du même coup $P3$ est soutenue. Dans ce cas, il n'est pas nécessaire de trouver une autre action pour le soutien de $P3$.

D'autre part, le soutien d'un ensemble de propositions peut amener différents groupes de propositions à être soutenus dans le niveau considéré après. Le schéma en bas à droite présente ces différents groupes dans l'ordre des combinaisons d'actions auxquelles ils correspondent. Certains de ces groupes peuvent mener à des impasses (suivant le reste du graphe de planification - non représenté ici).

Dans la première version de `Graphplan`, cette phase de recherche arrière est basique. En particulier, les retours arrières sont chronologiques. Cela signifie qu'en cas d'échec, la première décision à être remise en cause est la dernière à avoir été prise. Malheureusement, la décision "fautive" peut avoir été prise avant. Toutes les remises en cause intermédiaires en remontant chronologiquement à cette décision malheureuse sont inutiles.

Si l'alternance entre construction du graphe et recherche d'un plan se poursuit sans succès jusqu'à atteindre la saturation du graphe de planification, alors il est certain que le problème traité est sans solution. Les démonstrations pertinentes sont développées dans [6].

Définition : *Graphe saturé*

Lorsque deux itérations consécutives génèrent des niveaux identiques - mêmes actions, mêmes propositions, même relations de mutuelle exclusion, et qu'aucun nouvel ensemble de propositions n'est recensé comme échec, le graphe est dit *saturé*.

4.1.5 Améliorations des performances de `Graphplan`

Plusieurs améliorations algorithmiques ont été proposées :

- soit pour la phase de construction du graphe de planification ;
- soit pour la phase de recherche et d'extraction d'un plan.

Dans [71], les relations de mutuelle exclusion sont relaxées en des relations *d'autorisation*.

Dans [44], une implémentation optimisée du graphe de planification lui-même est proposée. Elle est mise en œuvre dans le planificateur `STAN`. Elle se base sur des propriétés de récursivité du graphe. Pour tout niveau propositionnel n, les propositions présentes sont celles qui étaient déjà présentes dans le niveau propositionnel précédent, plus d'éventuelles nouvelles propositions amenées par le dernier niveau d'actions. En d'autre termes, à partir du moment où une proposition est introduite dans un niveau du graphe, elle sera présente dans tous les niveaux suivants. Il en va de même pour les actions. Ainsi, plutôt que de construire explicitement le graphe en niveaux, il suffit de maintenir une liste des propositions et des actions avec leur premier niveau d'apparition.

Dans [37], Kambhampati se focalise sur la phase de recherche et d'extraction d'un plan. Il montre qu'elle peut être vue comme un problème de satisfaction de contraintes (*CSP*). Les variables y sont les propositions et les valeurs sont les actions pouvant éventuellement les supporter. La particularité de ce *CSP* est que toutes les variables ne nécessitent pas d'assignation. Certaines sont inactives et peuvent devenir actives au cours de la résolution (Kambhampati parle de *CSP dynamique*).

Au début de la résolution, les seules variables actives sont les objectifs. Suivant les valeurs qui leurs sont assignées, d'autres variables s'activent. Il s'agit en fait des préconditions des actions sélectionnées.

Reformuler cette recherche arrière en termes d'un problème de satisfaction de contraintes permet de mettre à profit tous les outils de ce domaine. En particulier, [37] présente des façons plus élaborées d'ordonner les variables à traiter, de choisir la décision à remettre en cause en cas d'échec, ou encore de propager certaines contraintes pour s'apercevoir plus tôt d'une voie sans issue. Les gains d'efficacité qui en découlent sont importants.

4.2 Efficacité de la planification disjonctive : point de vue de la planification heuristique

Nous l'avons vu : `Graphplan` construit un graphe de planification qui encode une approximation de l'atteignabilité des états. Cette approximation permet ensuite de guider la recherche d'un plan à proprement parler. En particulier, elle indique quand il semble raisonnable de lancer une recherche.

Ces notions d'approximation et de guidage de la recherche font penser à une heuristique. Ainsi, Bonet et Geffner défendent la vision de `Graphplan` comme un planificateur heuristique ([8]). Dans ce cadre, l'efficacité de `Graphplan` provient essentiellement de la précision de son heuristique et de la manière dont il en tire parti.

Pour Bonet et Geffner, l'heuristique de `Graphplan` renvoie le premier niveau du graphe de planification pour lequel toutes les propositions de l'état considéré sont présentes, non mutuellement exclusives, et non répertoriées comme un échec. Les auteurs la présentent comme une version plus fine de leur h_{max}. En d'autres termes, le graphe construit par `Graphplan` a pour objet de donner une borne minorante de la distance d'un état s à l'état initial.

La recherche peut alors être vue comme un algorithme de type IDA^* [40]. Il s'agit d'une recherche de type A* (cf. section A.3.3 de l'annexe 1) adaptée pour une mise en œuvre en profondeur d'abord itérée (cf. section A.2.2 de l'annexe 1).

Ici la première valeur de la profondeur maximale est donnée par le premier niveau dans lequel apparaissent les objectifs, non mutuellement exclusifs. Puisque la recherche arrière se fait dans le graphe construit, il est clair qu'elle ne va pas aller à des profondeurs plus grandes.

Notons que cette recherche met elle-même à jour l'heuristique. En effet, lorsqu'un ensemble de propositions ne peut être soutenu, il est mémorisé comme un échec.

Si aucun plan solution n'est trouvé, la recherche reprend avec une profondeur maximale incrémentée de 1. Dans `Graphplan` cela prend la forme de l'ajout d'un niveau au graphe et de la réitération de la recherche arrière.

Par rapport à `HSPr` par exemple, `Graphplan` ne mène pas sa recherche dans le même espace. En effet, tous deux sont des espaces de régression (depuis les buts vers l'état initial), mais le second utilise des sortes de macro-actions dues au fait que l'exécution d'actions en parallèle est autorisée. Celles-ci sont composées d'actions primitives compatibles (c'est-à-dire qui peuvent être exécutéees en même temps). Dans cet espace, l'heuristique de `Graphplan` reste admissible, tout en étant plus proche de la valeur exacte.

Dans le cadre de cette vision heuristique de `Graphplan`, nous allons maintenant présenter deux planificateurs laissés de côté lors de la présentation des autres approches heuristiques (section 2.3) : `FF` et `AltAlt`. Ce sont deux planificateurs ouvertement heuristiques. L'intérêt de les présenter ici est que le calcul de leurs heuristiques est lié à `Graphplan`.

4.2.1 Heuristiques liées à Graphplan

FF reprend l'idée de HSP d'utiliser un problème relaxé pour calculer son heuristique. Il s'agit du problème à traiter dans lequel les effets négatifs de toutes les actions ont été supprimés. HSP ne pouvant résoudre ce problème en un temps raisonnable pour extraire l'heuristique avait par ailleurs supposé tous les objectifs indépendants, approchant la solution au problème relaxé.

FF évite cette approximation et calcule une solution exacte du problème relaxé. Pour ce faire il utilise simplement Graphplan. Si les actions n'ont aucun effet négatif, Graphplan ne trouve aucune relation de mutuelle exclusion. S'il n'y a pas de relation de mutuelle exclusion, la recherche arrière se déroule sans retour-arrière (sans *backtracks*). Ainsi, le graphe de planification est développé de niveau en niveau jusqu'à l'apparition de tous les objectifs. La recherche arrière est alors lancée. Une première sélection d'actions pour supporter les objectifs est formée - elle est nécessairement valide dans le problème relaxé. Au niveau précédent, de la même manière, le premier choix est le bon (du fait de l'absence de relation de mutuelle exclusion). Le processus continue sans jamais devoir revenir en arrière jusqu'au premier niveau.

Une solution exacte au problème relaxé peut donc être trouvée rapidement. Elle est parallèle, c'est-à-dire que chaque étape du plan peut contenir plusieurs actions. Pour guider sa recherche, FF utilise comme heuristique le nombre d'actions à exécuter pour atteindre l'objectif plutôt que le nombre d'étapes ou de niveaux à enchaîner pour arriver à l'objectif. Cette heuristique n'est pas admissible.

AltAlt utilise différemment Graphplan. Il en exploite davantage le graphe de planification mais délaisse totalement la phase de recherche arrière. L'idée est de produire un graphe de planification - une fois - puis d'en extraire les valeurs heuristiques nécessaires à un algorithme de recherche adapté de HSPr.

Pour un problème donné, le graphe de planification est construit de la même façon que Graphplan à quelques exceptions près. D'abord, une relation de mutuelle exclusion supplémentaire est introduite entre chaque paire d'actions d'un même niveau. Ceci a pour but de linéariser les plans encodés dans le graphe en les rendant séquentiels. En effet, HSPr a besoin d'une heuristique pour plans séquentiels.

Ensuite le graphe est construit jusqu'à ce qu'il soit saturé. Une version de AltAlt relaxe cette contrainte et permet de ne construire le graphe que jusqu'au premier niveau contenant les objectifs non mutuellement exclusifs.

A partir de ce graphe plusieurs heuristiques peuvent être calculées. [58] discutent ces aspects en détail. En particulier, le premier niveau auquel apparaît une proposition p est une borne inférieure de la distance entre p et l'état initial. Pour un ensemble de propositions $\{p_1,p_2,...,p_n\}$, il est possible de définir l'heuristique h_{sum} comme étant la somme des premiers niveaux d'apparition de chaque proposition.

h_{sum} fait la même hypothèse d'indépendance des propositions que h_{add} (utilisée par le HSPr original - cf. 2.3.1). Cependant, elle est basée sur le problème réel - en prenant en compte les effets négatifs - et non sur un problème relaxé comme c'est le cas de h_{add}.

Pour réduire encore l'erreur de h_{sum}, les auteurs préfèrent utiliser pour AltAlt une heuristique proche de celle de FF. Il s'agit d'extraire un plan sans prendre en compte les relations de mutuelle exclusion (revenant ainsi à un comportement proche de celui de FF), puis de corriger a posteriori cette valeur pour prendre en compte dans une certaine mesure les interactions

négatives entre les sous-buts. A cette fin, pour chaque paire d'objectifs p et q, est calculé le nombre de niveaux entre celui où p et q sont présents ensemble pour la première fois, et celui où ils sont présents sans être mutuellement exclusifs. Cette valeur est notée $\delta(p,q)$. La valeur maximale de cette différence est retranchée à la longueur du plan pour le problème relaxé :

$$h_{AdjSum2M}(S) = length(RelaxedPlan(S)) + max_{p,a \in S}\delta(p,q) \tag{4.1}$$

$h_{AdjSum2M}(S)$ est la notation utilisée par les auteurs pour *Heuristique ajustée.*

Cette approche se révèle meilleure que *HSPr*. Il faut noter aussi que la construction du graphe est laissée à *STAN* qui optimise l'espace mémoire utilisé pour construire le graphe.

4.3 Efficacité de la planification disjonctive : point de vue de la planification par raffinage

Dans [38], l'explication de l'efficacité de `Graphplan` par Kambhampati et al. ne fait pas référence à une interprétation en termes d'heuristiques. Elle place `Graphplan` dans un cadre plus général : la *planification par raffinage* [36]. Le planificateur étudié devient alors une instance particulière d'un schéma de planification générique. Kambhampati et al. met en avant la structuration particulière de son espace de recherche par rapport aux instances existant auparavant pour expliquer son gain d'efficacité sur ces dernières.

Principe de la *Planification par raffinage*

> Partant de l'ensemble de toutes les séquences d'actions possibles, un planificateur a pour tâche de le réduire petit à petit jusqu'à obtenir l'ensemble de toutes les séquences qui sont des solutions. La réduction progressive de l'ensemble des séquences considérées introduit la notion de raffinages successifs.

Plutôt que de toujours travailler sur un seul ensemble de séquences d'actions possibles, il peut être préféré de diviser le travail de planification en branches où des sous-ensembles de séquences d'actions sont traités indépendamment. Cette notion de *division* est la deuxième action de base de cette vision de la planification.

Tout planificateur se définit alors en termes d'instances particulières des procédures de raffinage et de division. L'algorithme 1, ci-dessous, présente l'algorithme générique de la planification par raffinage.

D'une manière générale, remarquons que la croissance d'un plan partiel va de paire avec la réduction de l'ensemble des plans associés. En effet, les actions déjà sélectionnées et leurs positions sont autant de contraintes pour le plan. Plus il y en a, moins il y a de plans susceptibles de les satisfaire.

Dans le cas de *FSS*, le raffinage se fait par croissance de préfixes de plans. En effet, les actions du plan partiel sont ajoutées une à une en partant de la première et en les plaçant de manière contigüe. A partir d'un préfixe donné, c'est-à-dire à partir d'un ensemble de plans potentiels donné, tous les préfixes obtenus par ajout d'une seule action (tous les ensembles de plans potentiels réduits par ajout d'une contrainte) sont traités indépendamment. Il est question de *division totale* (ou *morcellement total*).

De manière analogue, il est possible de décrire les approches de type *BSS* ou dans l'espace des plans. `Graphplan`, est présenté comme appliquant la même strategie de raffinage que *FSS*, mais avec une stratégie de division différente. En effet, la progression de la planification se

Algorithme 1: Algorithme générique de planification par raffinages successifs tel qu'il est présenté dans [36]. Un *candidat minimal* d'un ensemble de plans est un plan constitué des seules actions du plan partiel considéré.

```
Raffiner(P : ensemble de plans)
si P = ∅ alors
    retourner échec
sinon
    si un candidat minimal de P est solution alors
        retourner solution
    sinon
        Sélectionner une stratégie de raffinage R
        P' ← Appliquer R à P
        Morceler P' en k ensembles de plans
        Simplifier les ensembles de plans par propagation de contraintes
        Choisir de manière non déterministe un ensemble de plans P'_i
        Appeler Raffiner(P'_i)
```

fait par croissance de préfixes de plans, cependant tous les préfixes possibles en ajoutant une action à un préfixe donné sont gardés ensembles. Les ensembles de plans associés ne sont pas séparés pour être traités. Il est alors question de *division nulle* (ou *morcellement nul*).

L'argumentation de Kambhampati et al. se focalise sur le fait qu'en l'absence de division, `Graphplan` s'affranchit des duplications d'ensembles - et donc de travail - que la division du travail de planification en branches de recherche indépendantes ne manque pas d'introduire. Nous avons vu que *FSS* peut "s'étouffer" dans un arbre truffé de branches inutilement reconstruites. Ces branches sont le fruit de la division appliquée. A l'opposé, `Graphplan` n'introduit aucune division et résoud précisément cette difficulté.

A ce sujet, les figures 2.1 (page 32) et 4.9 peuvent être utilement comparées. La première montre l'arbre de planification développé par *FSS* pour un problème de transport de balles par Robby. La seconde montre le graphe de planification généré par `Graphplan` pour le même problème. Le premier niveau est identique à la racine de l'arbre. Le premier niveau d'actions rassemble toutes les actions introduites sur les premiers arcs de l'arbre. Le deuxième niveau propositionnel est l'union des propositions constituant les états des nœuds de profondeur 2. Si `Graphplan` ne se contentait pas des relations binaire de mutuelle exclusion mais calculait aussi toutes les autres, alors chaque niveau n d'actions ou de propositions serait l'union des actions ou des propositions présentes dans l'arbre en profondeur n.

Alors que dans *FSS* les propositions sont regroupées par états, dans `Graphplan`, les états sont regroupés par proposition. L'action $(MoveZ_1Z_2)$ a besoin de trois préconditions. Robby doit être en Z_1, les zones Z_1 et Z_2 doivent être connectées, et enfin Z_2 doit être libre. Les positions des balles, l'accessibilité des autres salles, leur occupation éventuelle, etc., ne sont pas pertinentes ici. Il s'en suit que de nombreux états peuvent être valides pour l'exécution de cette action. `Graphplan` les considèrent tous ensemble, en une fois, au travers de leur ensemble décrit par les trois propositions des préconditions. *FSS* est obligé de considérer chaque état indépendamment, alors même que la seule chose intéressante dans ces états est la présence des trois préconditions. C'est cette différence fondamentale qui explique l'efficacité élevée de

`Graphplan` par rapport aux autres planificateurs de l'époque.

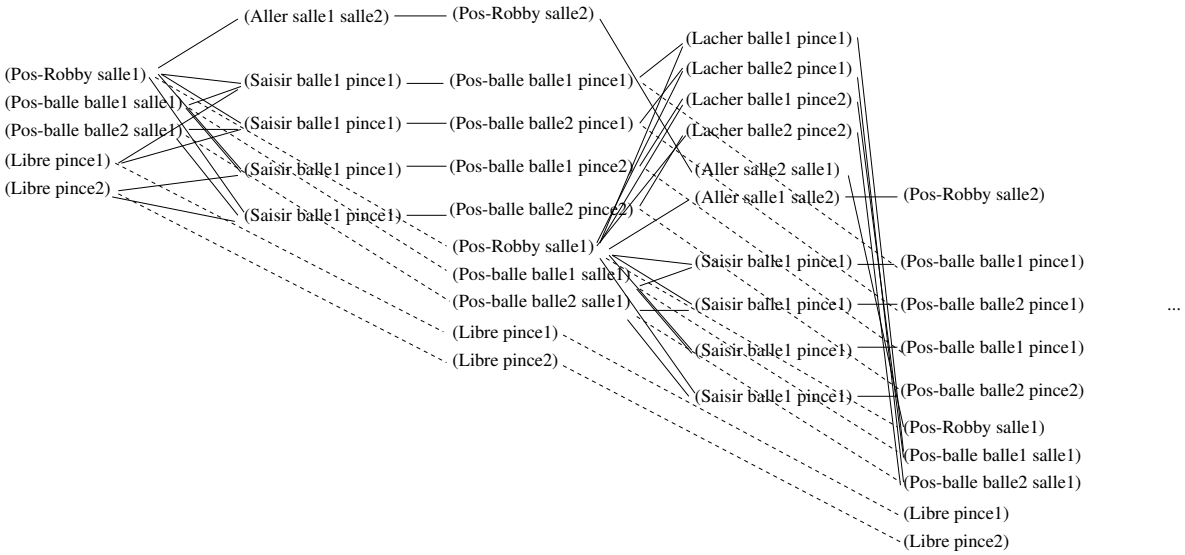


FIG. 4.9 – *Voici le début du graphe de planification construit par* ***Graphplan*** *pour le problème de Robby que nous utilisons comme exemple. Les actions* NoOp *sont figurées en pointillés.*

4.4 Extensions à l'incertain et à l'optimisation

`Graphplan` a inspiré de nombreux travaux. En particulier, les gains d'efficacité qui l'ont accompagné - pour le problème classique de planification - ont laissé espérer des gains substanciels pour les problèmes de planification non classiques. Quelques explorations d'adaptation de `Graphplan` pour appliquer la planification disjonctive à l'optimisation et à la prise en compte de l'incertitude sont présentées ici.

4.4.1 Pgraphplan

`Pgraphplan` [7] a été développé en 1998. Il vise à produire des plans contingents soit maximisant leurs chances de réussite, soit minimisant un coût - une durée de trajet en voiture dans l'exemple présenté. La longueur du plan solution est fixée a priori.

Les états sont totalement observables, mais les actions sont probabilistes. Ainsi, les effets de chaque action peuvent être répartis en sous-ensembles correspondant aux diverses alternatives possibles. La construction du graphe se fait comme dans `Graphplan`, excepté que chaque arc entre une action et un effet indique avec quelle probabilité cet effet est produit et à quelle alternative il appartient (cf. figure 4.10).

Contrairement à `Graphplan`, la recherche d'une solution ne se fait pas des objectifs vers l'état initial mais à l'inverse. En effet, une recherche arrière est rendue très difficile par la nécessité de calculer les probabilités d'occurrence des propositions considérées. Celles-ci dépendent de ce qu'il s'est passé avant. Or, il n'est pas possible de prendre en compte les probabilités d'occurrence d'effets du début d'un plan en construisant ce plan par la fin! Il s'en

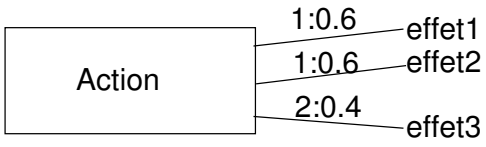


FIG. 4.10 – *Dans cet exemple, l'exécution de l'action peut avoir deux conséquences différentes. Soit elle provoque les effets* 1 *et* 2*. Cette première possibilité a une probailité de* 0.6 *d'arriver. Soit elle provoque l'effet* 3 *- avec une probabilité de* 0.4

suit une série de va-et-vient coûteux. La figure 4.11 montre ces difficultés sur un exemple. Notons que le planificateur présenté dans [29] a fait le choix d'une recherche arrière. Il prend en compte des actions non déterministes, et des actions d'observations, dans une modélisation de l'incertitude utilisant la théorie des possibilités [16]. Ses performances souffrent précisément des va-et-vient que nous venons d'évoquer.

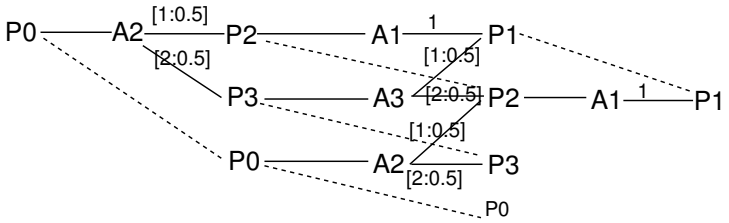


FIG. 4.11 – *Voici un exemple de graphe produit par* **`Pgraphplan`***. Il est représenté partiellement. Les pointills correspondent à des actions* NoOp*. Le but a soutenir - à droite - est* $P1$*. Nous allons montrer qu'une recherche arrière peut entraîner des va-et-vient. Choisissons* NoOp *comme soutien. Il faut ensuite soutenir à nouveau* $P1$ *au niveau précédent.* $A1$ *le soutien avec une probabilité de* 1*. Il faut maintenant soutenir* $P2$ *au niveau précédent.* $P2$ *n'est soutenu que par* $A2$ *avec une probabilité de* 0.5*. Cela signifie que* $P1$ *que nous croyions soutenue avec une probabilité de* 1 *au dernier niveau ne l'est qu'avec une probabilité de* 0.5*. Il faut absolument traiter les autres cas. Pour cela, il faut repartir en avant.* $A2$ *peut aussi avoir pour effet* $P3$ *au deuxième niveau de propositions.* $P3$ *est précondition de* $A3$ *qui soutient* $P1$ *à* 0.5 *au troisième niveau. Le soutien de* $P1$ *est donc remonté à* 0.75 *au troisième niveau.* $A3$ *soutient alternativement soit* $P1$ *soit* $P2$ *à* 0.5 *au troisième niveau.* $P2$ *permet de déclencher* $A1$ *et d'obtenir* $P1$*. Le soutien de* $P1$ *est revenu à* 1 *au dernier niveau. Nous pouvons revenir aux préconditions de* $A2$ *et chercher un soutien pour* $P0$*.*

Malheureusement, en choisissant une recherche avant dans le graphe de planification pour `Pgraphplan`, les auteurs de [7] ont sacrifié les avantages des contraintes propagées lors de la construction du graphe. En effet, celles-ci étant propagées vers l'avant, elles favorisent les recherches arrière. Prenons l'exemple des relations de mutuelle exclusion. L'apparition de l'une d'entre-elles à un niveau n permet de contraindre tous les niveaux qui suivent. Une recherche partant des objectifs peut très bien être interrompue à un niveau $n+10$ par exemple, du fait des contraintes propagées depuis le niveau n. Au contraire, une recherche avant ne s'arrêtera pas avant le niveau n à cause de cette relation de mutuelle exclusion. Rien n'a été propagé

pour indiquer que cette voie est sans issue.

Conscients de ce problème, les auteurs ont doté leur planificateur de mécanismes de propagation arrière. L'un retire les branches (et récursivement les nœuds) qui ne mène pas aux objectifs. Cela réduit la largeur du graphe. L'autre propage sur les propositions des valeurs heuristiques de leurs valeurs d'utilité. Cela permet de ne pas explorer des branches clairement moins bonnes que d'autres.

Malgrè cela, la perte de performance par rapport à Graphplan est grande. [7] contient d'ailleurs une comparaison avec `Graphplan` sur des domaines déterministes pour isoler de la perte liée à la recherche avant, la difficulté plus grande des problèmes probabilistes.

4.4.2 DT-Graphplan

`DT-Graphplan` [61] a des objectifs proches de ceux de `Pgraphplan`. Considérant des actions probabilistes, il vise à générer un plan permettant d'atteindre une série d'objectifs avec une valeur d'utilité supérieure à un certain seuil. Contrairement à `Pgraphplan`, il ne recherche pas de plan contingent (cf. définition section 1.5.3).

La différence majeure dans sa manière de fonctionner avec `Pgraphplan` est que le graphe qu'il construit contient diverses occurrences d'une même proposition si elles correspondent à des valeurs d'utilité ou d'incertitude différentes (cf. figure 4.12). Ainsi, la probabilité d'occurrence d'une proposition est calculée pendant la construction du graphe, en avant. Ce calcul étant fait avant la recherche d'un plan, il n'y a plus d'obstacle à ce que cette recherche se fasse en arrière, comme dans `Graphplan`. La recherche d'un plan retrouve ainsi le bénéfice de la propagation de contraintes réalisée pendant la construction du graphe. En contre-partie, bien-sûr, le graphe contient beaucoup plus de nœuds par niveau. Il est donc plus lourd d'y effectuer une recherche.

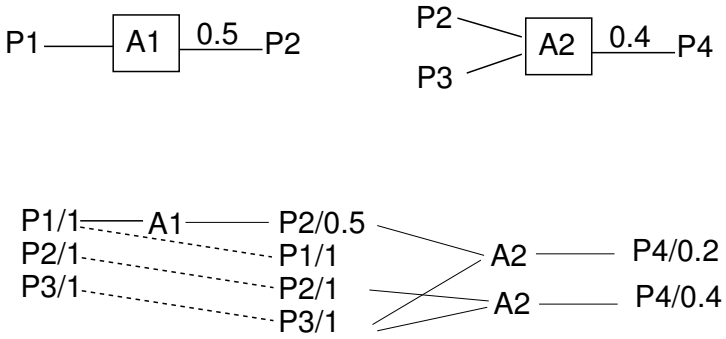


FIG. 4.12 – *Voici un détail d'un graphe de planification de DT-Graphplan. Deux actions sont considérées.* $A1$ *produit* $P2$ *avec une probabilité de* 0.5. $A2$ *produit* $P4$ *avec une probabilité de* 0.4. *Les actions* NoOp *sont représentées en pointillés. Au deuxième niveau propositionnel,* $P2$ *est présente deux fois (avec des probabilités différentes). En conséquence, l'action* $A2$ *est introduite deux fois au niveau suivant, générant deux fois* $P4$*, avec des probabilités différentes. A titre d'exemple, voici comment calculer la probabilité de la première* $P4$ *(celle du haut). L'ensemble des préconditions de* $A2$ *est atteignable avec une probabilité de* 0.5 *(hypothèse d'indépendance,* $0.5 * 1 = 0.5$*).* $A2$ *produisant son effet avec une probabilité de* 0.4*, nous obtenons* $0.2 = 0.5 * 0.4$.

Pour rendre cette approche possible, deux simplifications ont dû être concédées.

D'une part, les occurrences des propositions sont considérées indépendantes du point de vue probabiliste. Ainsi, la probabilité qu'une action puisse être déclenchée à un niveau donné est le produit des probabilités d'occurrence de ses préconditions. D'autre part, les valeurs d'utilité sont attribuées aux caractéristiques des états plutôt qu'aux états eux-mêmes. C'est bien entendu une condition très restrictive.

Prenons l'exemple d'un skieur. Avoir des skis à Paris ne lui apporte pas grand-chose. Etre dans une station des Alpes sans ski n'est pas mieux. La valeur d'utilité va devenir non nulle s'il se retrouve dans un état où il est dans une station des Alpes **et** il a des skis à sa disposition. A partir de là elle peut varier suivant la qualité de la neige, l'ensoleillement, etc. . Cependant chacun de ces éléments pris indépendemment - comme c'est le cas dans `DT-Graphplan` - n'a aucune valeur intrinsèque.

4.4.3 Conformant Graphplan et Sensory Graphplan

`Conformant Graphplan` [75] se place dans un cadre sensiblement différent de ceux évoqués précédemment. L'état initial est incertain, ainsi que les effets des actions. Cependant aucune probabilité n'entre en jeu. Ainsi les chances d'occurence de tel ou tel effet ne sont pas quantifiées. Le planificateur recherche un plan conformant (cf. définition section 1.5.2).

Dire que l'état initial est incertain revient à dire que plusieurs états initiaux différents sont possibles. `Conformant Graphplan` considère ces différents états un à un, dupliquant les éventuelles propositions qu'ils ont en commun. Il va développer un graphe à la manière de `Graphplan` pour chacun d'entre-eux. Si des actions à effets incertains sont utilisées introduisant de nouveaux mondes possibles, le graphe se subdivise en autant de sous-graphes ou branches possibles.

La recherche arrière est lancée dès que les objectifs apparaissent dans tous les mondes considérés (c-à-d. dans tous les sous-graphes), sans être jamais mutuellement exclusifs. La recherche progresse niveau par niveau en prenant en compte tous les mondes parallèles possibles. Quand une action semble convenir dans un monde, il faut vérifier qu'elle serait aussi compatible dans les autres mondes. En effet, gardons à l'esprit que lors de l'exécution, l'agent n'aura aucun moyen de savoir dans quel monde il se trouve effectivement. Si l'action ne convient pas dans un monde, deux solutions sont possibles. Soit cette action est abandonnée et une autre est sélectionnée à sa place. Soit le reste du plan - concernant les niveaux précédents - est construit de telle manière qu'il assure que l'agent ne sera pas dans le monde problématique au moment de l'exécution de l'action considérée. Il est alors question de *confrontation*. Si par exemple un médicament à un effet négatif sur un patient lorsque celui-ci est deshydraté, il suffit de le faire boire au préalable pour s'assurer que cet effet secondaire ne pourra pas survenir.

Dans [74], `Conformant Graphplan` est augmenté de la prise en compte d'actions de perception pour donner `Sensory Graphplan`. Ce planificateur fonctionne de la même manière : générant plusieurs graphes indépendants. Cependant, bien que l'état initial puisse être incertain, et que les effets des actions sont conditionnés à l'état courant, les actions elles-mêmes sont déterministes. Cette fois, en cas de conflit entre plusieurs mondes, en plus de l'abandon de l'action et de la confrontation, une solution perceptive est possible. Le planificateur peut essayer d'inclure dans le reste du plan une action de perception lui permettant de distinguer les mondes qui posent problème. Le plan généré est alors un plan contingent.

4.5 Conclusion

La planification disjonctive, au travers de `Graphplan`, a eu un impact important sur la planification d'actions. Cependant, elle ne s'éloigne pas des sources classiques de la planification en exploitant au mieux les représentations en intension sans leur superposer de structuration à base d'état.

Cette approche disjonctive a aussi une influence sur la planification heuristique.

Dans le cadre plus spécifique de nos travaux, la planification disjonctive à une place de premier plan pour deux raisons. D'une part, elle manipule explicitement des ensembles d'états. D'autre part, elle sépare explicitement les étapes de construction de l'espace de recherche, et de recherche d'un plan dans ces espaces. En outre, l'extension de la planification disjonctive au delà du problème classique montre la nécessité que ces deux phases soit adaptées l'une à l'autre.

Chapitre 5

Synthèse de la partie 1

Les chapitres précédents ont présenté de nombreuses approches différentes de la planification d'actions. Elles forment bien-sûr un tout cohérent. Une typologie basée sur la distinction entre espace d'états et espace des plans ne nous semble pas adaptée. Nous en proposons une autre qui s'appuie sur les caractéristiques disjonctives des approches. Enfin, nous soulignons les conséquences de ces différences sur les la structuration de l'espace de recherche, laquelle joue un rôle fondamental en planification d'actions.

5.1 Planification disjonctive, espace d'états et espace des plans

Une typologie souvent utilisée distingue la planification dans l'espace des états (chapitre 2) et la planification dans l'espace des plans (chapitre 3). Elle semble ne pas être adaptée à la planification disjonctive.

Certes, il est facile de montrer qu'un graphe de planification du type de celui de `Graphplan` peut s'obtenir à partir de l'arbre de *FSS* en unifiant systématiquement les nœuds de même profondeur. Il s'agit en fait du graphe que construirait un `Graphplan` qui tiendrait compte de toutes les mutuelles exclusions. Nous l'avons évoqué en section 4.3 (page 70).

Cette construction tend à placer `Graphplan` dans la catégorie des planificateurs dans l'espace des états. Toutefois, une construction analogue peut s'accomplir à partir d'un arbre de planification dans l'espace des plans.

En effet, reprenons l'arbre de la figure 3.4 (page 52). Rappelons, qu'il s'agit de planification dans l'espace des plans, avec des contraintes d'insertion des actions à la suite de celles déjà introduites, de manière contigüe. Si les noeuds de chaque profondeur sont unifiés entre eux, la séquence de la figure 4.9 est obtenue. Tout noeud de profondeur n contient alors le graphe de planification d'un "`Graphplan` parfait" à la n-ième itération.

Par ailleurs, `Graphplan` n'utilise jamais de référence explicite aux états. Au contraire, il manipule directement les propositions qui les composent - comme en planification dans l'espace des plans.

Dès lors, `Graphplan` ressemble beaucoup à un planificateur dans l'espace des plans. Cependant, il ne raisonne pas en termes de modifications de plans, mais bien en termes de zone atteignable dans l'espace d'états.

En plus de l'importance de la planification disjonctive dans le cadre de nos travaux, c'est aussi pour ces raisons que nous avons présenté la planification disjonctive en dehors des approches dans l'espace des états d'une part et dans l'espace des plans d'autre part.

5.2 Une typologie adaptée à l'ensemble des approches de planification

Faisons abstraction de l'utilisation d'un espace d'états ou d'un espace des plans, puisque baser une typologie sur cette distinction nous a mené à des ambigüités. La génération d'un plan nécessite deux fonctionnalités primordiales :

- Sélectionner les actions à introduire dans le plan
- Ordonner les actions du plan

Lors de la conception d'un planificateur, un paramètre essentiel est le moment où il prendra ses décisions. Plus précisément, il est question du moment où ses choix seront scellés. Par exemple, la caractéristique principale des approches de moindre engagement en planification dans l'espace des plans est de différer le plus possible les choix concernant l'ordonnancement des actions.

Nous obtenons une typologie à deux dimensions : sélection et ordonnancement. Chacune d'entre elles recouvre deux valeurs : choix immédiat et choix différé (cf. figure 5.1).

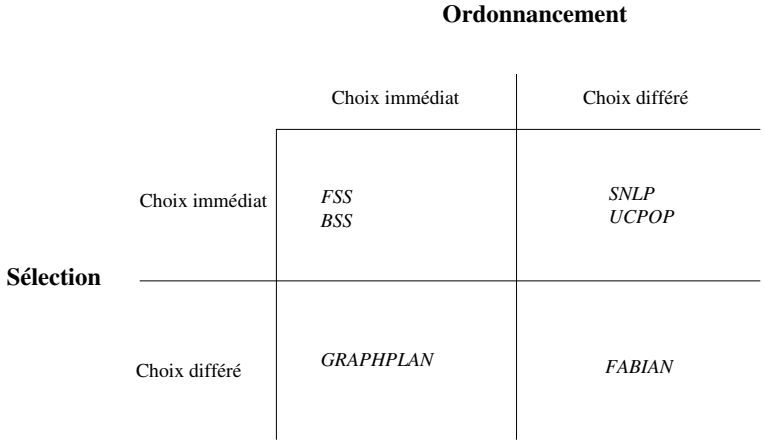


FIG. 5.1 – *Une typologie des planificateurs basé sur l'immédiateté de leurs choix. En d'autres termes c'est une typologie basée sur la capacité des planificateurs à gérer des disjonctions de possibilités soit en matière de sélection d'actions, soit en matière d'ordonnancement d'actions.*

Les approches arborescentes dans l'espace des états scellent leurs choix immédiatement, tant pour la sélection des actions que pour leur ordonnancement. C'est aussi le cas de l'approche de planification dans l'espace des plans que nous avons utilisée pour reconstruire le graphe d'un "`Graphplan` parfait".

Les approches de moindre engagement dans l'espace des plans, nous l'avons vu, diffèrent le choix de la position exacte de l'action introduite. En revanche, elles maintiennent l'immédiateté de la sélection de l'action qui sera dans le plan.

La planification disjonctive - au travers de `Graphplan` - est dans le coin opposé. Le choix de la position des actions est scellé immédiatement. En effet, les instances d'actions introduites lors de la deuxième itération ne pourront être utilisées que dans l'étape 2 du plan. Par contre, la sélection de l'action à introduire dans le plan parmi toutes celles introduites lors d'une itération est différée.

En fait, `Graphplan` et les planificateurs à ordre partiel ne sont pas plus ou moins de moindre engagement les uns que les autres. Cependant ils le sont sur des paramètres différents. `Graphplan` pourrait être qualifié de moindre engagement sur la sélection des actions. De la même manière, les planificateurs à ordre partiel manipulent des représentations disjonctives des ordonnancements possibles des actions.

Dans ce dernier cas, les divers ordonnancements possibles sont considérés implicitement. Au contraire, `Graphplan` explicite les divers choix d'actions possibles.

Reste une dernière catégorie : les approches différant à la fois leurs choix concernant la sélection des actions et le positionnement de ces dernières. Une expérimentation en ce domaine est le planificateur `Fabian`, présenté dans [26]. Il est basé sur un planificateur à ordre partiel. Un pré-traitement agglomère toutes les actions partageant un même effet en une *action abstraite*. Par la suite, lorsqu'une proposition doit être soutenue, c'est l'action abstraite correspondante qui est introduite. Ainsi, il est évité de considérer individuellement tous les cas correspondant aux différentes actions élémentaires possibles.

Les actions abstraites sont construites en prenant l'intersection des préconditions des actions élémentaires et l'union de leurs effets. Seuls les effets présents dans toutes les actions élémentaires sont considérés pour détecter les conflits. De plus, si un effet utilisé n'est pas présent dans toutes les actions élémentaires, alors l'action abstraite est modifiée pour ne garder que les actions élémentaires pertinentes.

L'étude expérimentale présentée dans [26] montre quelques gains de performance sur le planificateur à ordre partiel `snlp`. Toutefois, l'amélioration n'est pas très grande et rappelons que `Graphplan` surpassait largement les planificateurs à ordre partiel.

Des travaux récents s'intègrent dans cette typologie, comme des efforts pour explorer l'une ou l'autre des deux dimensions. C'est le cas par exemple du développement du planificateur `BBG` (cf. [32]). Basé sur `Graphplan`, ce planificateur scèlle cependant dès la construction du graphe de planification des choix quant à la présence ou non d'une action à un niveau donné. Ces décisions ne sont pas prises systématiquement. En conséquence, `BBG` se situe à cheval sur les deux cases de la colonne de gauche : une partie des choix de sélection sont scellés au plus tôt, mais pas tous.

5.3 Diversité et importance des structurations possibles de l'espace de recherche

En toute approche de planification deux phases peuvent être distinguées :

1. l'une de construction de l'espace de recherche ;
2. l'autre de recherche et d'extraction éventuelle d'un plan-solution dans cet espace.

Dans `Graphplan`, elles sont explicitement distinguées - construction du graphe d'une part, recherche arrière d'autre part - et alternent l'une l'autre. Dans les approches par chaînage arborescent, l'essentiel des efforts est concentré sur la construction de l'espace de recherche. La seconde phase en est réduite à un minimum trivial. En effet, dès qu'une feuille est compatible avec les objectifs, il est certain qu'une solution est présente dans l'espace. Il suffit d'extraire cette feuille si le travail s'est effectué dans l'espace des plans, ou le chemin - unique - entre la racine et cette feuille dans le cas d'un travail dans l'espace des états.

Ces deux phases ne sont bien évidemment pas indépendantes. Suivant la structure de l'espace de recherche, certaines recherches seront possibles ou non, facilitées ou rendues plus difficiles. Ces contraintes apparaissent clairement quand il s'agit de traiter des extensions du problème classique de planification comme celles concernant les incertitudes ou l'optimisation de critères.

En effet, la résolution de ce type de problèmes nécessite des informations supplémentaires à celles nécessaires à la résolution du problème classique. Ces informations sont parfois perdues quand la structuration de l'espace de recherche est optimisée pour le problème classique de planification.

Un aspect primordial de la construction de l'espace de recherche est de définir ce qui peut être distingué de ce qui ne peut pas l'être pendant la recherche. De manière duale, elle définit ce qui peut être abstrait de ce qui ne peut pas l'être.

Les recherches arborescentes dans l'espace d'états distinguent chaque état individuellement et ne peuvent pas en traiter plusieurs comme un tout. En revanche, les approches de type `Graphplan` construisent des espaces de recherche dans lesquels de larges ensembles d'états peuvent être manipulés, mais où la reconstruction des états individuels demande plus d'efforts.

Dans le cadre de l'espace des plans, les approches dites *de moindre engagement* distinguent précisément les différents ensembles d'actions pouvant servir de base à un plan. En revanche, elles distinguent le moins possible les différents ordres applicables à ces ensembles. Elles peuvent être opposées aux approches de type `Graphplan`, lesquelles distinguent clairement les différents séquencements possibles des actions en distinguant assez peu les différents ensembles d'actions susceptibles de générer des plans.

Ces notions de distinction et d'abstraction semblent se relier aux capacités de traitement de disjonctions d'éléments (actions, états, ou séquencements). Elles se retrouvent aussi à la source d'un compromis entre la précision de l'étude d'atteignabilité que constitue la construction de l'espace de recherche et la lourdeur de cette phase. Enfin, agissant directement sur l'organisation de l'information à disposition, elles sont au cœur de la planification et peuvent à ce titre faire le lien entre des approches classifiées dans l'espace d'états et d'autres classifiées dans l'espace des plans.

Pour ces raisons, nos travaux ce sont focalisés sur cette structuration de l'espace de recherche et sur ses conséquences sur la phase de recherche de solution. C'est l'objet du reste de ce manuscrit.

Deuxième partie

Morcellement de l'espace de recherche

Chapitre 6

Morcellement : présentation

La phase de construction de l'espace de recherche est une étude d'atteignabilité de l'espace d'états depuis l'état initial. Un planificateur traite un problème de planification en manipulant la représentation mise à disposition du modèle du problème considéré.

Il ne va pas *réellement* atteindre les différents états du modèle pour les tester. Un robot d'exploration largué sur le sol d'une planète inconnue doit en effet arpenter le terrain pour le connaître. Il passe d'un état à un autre au risque de glisser dans un ravin - état final de son exploration!

Un planificateur, lui, reste au niveau de l'information. Ainsi, il bénéficie d'une plus grande flexibilité pour réaliser cette étude d'atteignabilité. Il n'est contraint que par les possibilités de la représentation qu'il utilise.

Une représentation en intension offre une grande latitude dans la manipulation d'ensembles d'états. La manière dont les états sont regroupés en "paquets" pour être manipulés ensemble nous paraît être un critère primordial :

- de différenciation des approches actuelles de planification dans l'espace d'états ;
- de construction d'approches adaptées à des problèmes de planification spécifiques.

C'est la notion de ***morcellement***.

`FSS`, par exemple, réalise une étude d'atteignabilité rigoureusement exacte en restant collé à la notion d'état. Tel un simulateur, il déroule l'arbre des chemins possibles, d'état en état.

`Graphplan`, en revanche, manipule les ensembles d'états les plus larges possibles pour se faire rapidement une idée - certes moins précise - de ce qui est atteignable. Il déroule une séquence de grands ensembles d'états plutôt qu'un arbre d'états individuels.

Nous l'avons vu, cette approche est poussée encore plus loin par des planificateurs heuristiques récents qui relaxent certaines contraintes du modèle afin d'exploiter la représentation pour que cette étude d'atteignabilité approchée leur serve d'heuristique.

Nous pensons qu'une des causes des difficultés rencontrées par les planificateurs classiques pour traiter efficacement des problèmes non classiques est précisément l'inadéquation aux buts qu'ils poursuivent des paquets d'états qu'ils manipulent (et donc de la structure arborescente de leurs espaces de recherche).

Nous discutons au travers d'un exemple l'influence de la structuration arborescente de l'espace de recherche d'un planificateur sur la résolution d'un problème donné. Nous suggérons qu'une arborescence intermédiaire est pour cet exemple plus appropriée.

Ensuite, nous montrons, grâce à la formalisation de la notion de morcellement, que les structurations des espaces de recherche respectifs de `FSS` et `Graphplan` ne diffèrent effectivement que de par le morcellement mis en œuvre. Nous montrons du même coup qu'une grande variété d'autres approches sont possibles grâce à une grande flexibilité des stratégies de morcellement possibles.

L'enjeu est alors d'adapter le morcellement de l'espace de recherche aux spécificités du problème considéré pour le traiter plus efficacement.

L'idée est de réintroduire de l'arborescence dans l'espace de recherche de la planification disjonctive de manière à améliorer un compromis entre efficacité et précision de l'étude d'atteignabilité. Une telle structure arborescente permet de distinguer seulement ce qui mérite de l'être. Le reste est traité de manière disjontive au sein de chaque branche. Ainsi, chaque nœud ne correspond pas à un état mais à un ensemble d'états.

6.1 Objectif : contrôler la structuration arborescente de l'espace de recherche

La figure 6.1 présente au travers d'un exemple une vision intuitive de notre approche.

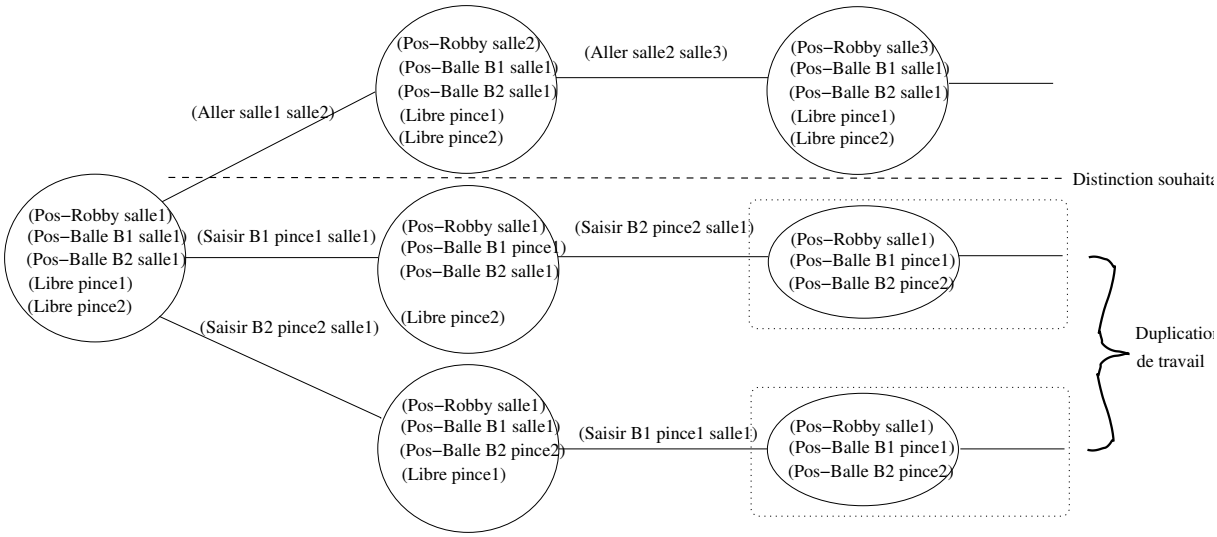


FIG. 6.1 – *Cet exemple vise à donner une idée intuitive de notre approche - bien qu'il ne corresponde pas à une réelle stratégie de branchement. L'arborescence mène ici à une duplication de travail - source d'inefficacité. Les deux branches du bas pourraient être traitées ensemble, sur le mode de la planification disjonctive. Par contre la branche du haut semble constituer une distinction "saine". A priori, elle pourrait être conservée. Nous obtiendrions alors une arborescence intermédiaire entre totale et nulle.*

Réintroduire de l'arborescence dans la planification disjonctive peut lui permettre en particulier d'appréhender des problèmes de planification non classiques. Par exemple, l'approche utilisée par `Conformant Graphplan` ou `Sensory Graphplan` (cf. section 4.4.3) pour générer respectivement des plans conformants et contingents, revient à construire une arborescence à deux branches. Chaque branche correspond à l'un des graphes générés - un pour chaque état initial possible.

Les difficultés rencontrées par `Pgraphplan` (cf. section 4.4.1) et `DT-Graphplan` (cf. section 4.4.2) proviennent dans les deux cas d'un manque d'arborescence en planification disjonctive. En effet, leurs problèmes sont dûs au fait que deux occurrences d'une même proposition peuvent avoir des valeurs d'utilité ou de probabilité différentes liées aux différentes séquences d'actions permettant de les obtenir.

Dans certains cas, ce n'est pas de deux occurrences d'une même proposition qu'il s'agit mais de deux occurrences d'un même ensemble de propositions (comme nous l'avons vu en évoquant les limitations de `DT-Graphplan`). Introduire une arborescence liée aux valeurs d'utilité ou de probabilités de ces ensembles de propositions permettrait de suivre le cheminement au sein de l'espace d'utilités (ou de probabilités), et en conséquence de ne pas perdre cette information cruciale pour la tâche d'optimisation considérée.

Inversement, si plusieurs états ont la même utilité, il est inutile (et donc inefficace) de les distinguer. Il est donc bien question encore une fois de distinguer des ensembles d'états et non des états. Nous suggérons une arborescence intermédiaire. Celle-ci permettrait de tirer parti d'une recherche arrière en optimisation dans des cas réalistes - en évitant les va-et-vient forçant `Pgraphplan` à utiliser une recherche avant ou les hypothèses simplificatrices probablement peu réalistes de `DT-Graphplan`.

De manière imagée (cf. figure 6.2), assimilons la planification à un chantier ayant pour but de construire des murs avec des motifs géométriques bleus. Des pans de murs déjà assemblés en briques colorées sont livrés. Certains ont des lignes horizontales jaunes et vertes, d'autres une diagonale bleue, etc. . Le schéma disjonctif à la `Graphplan` démonte ces pans de murs pour obtenir les briques elles-mêmes. C'est pratique puisqu'il est alors plus facile de se focaliser sur le positionnement des briques bleues.

Pourtant, à ce stade, plutôt que de tout construire brique par brique, il est sans doute plus efficace de reconstituer des pans de murs comportant déjà les figures dont est composée la fresque à construire. L'assemblement final de ces pans de murs sera aisé et comportera moins de risques d'erreurs. C'est ce que propose notre approche.

En quelque sorte, ce processus est similaire au comportement du *Group By* de SQL dans le domaine des bases de données. En effet, celui-ci permet de présenter les données différemment, en les groupant de manière plus adaptée au problème à résoudre.

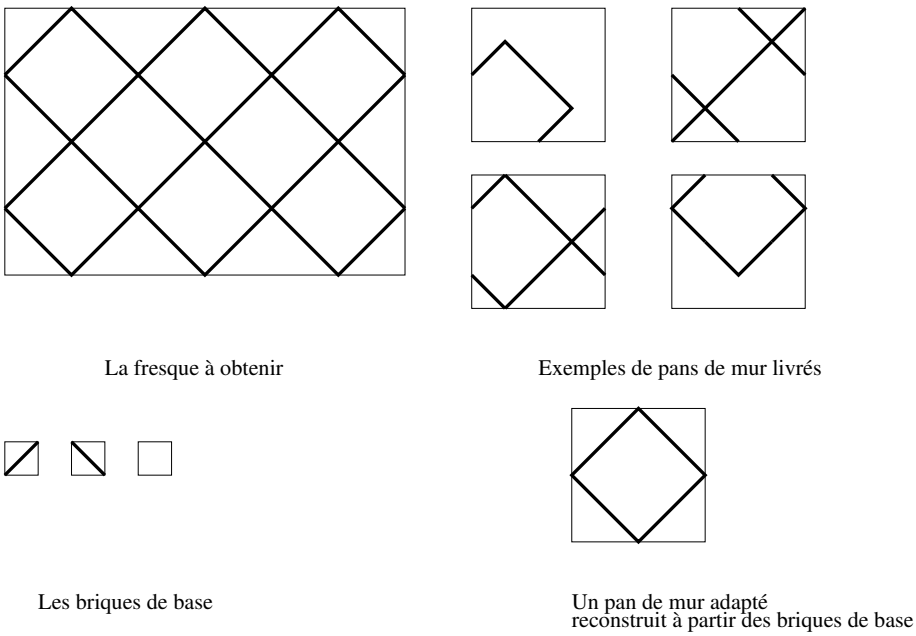


FIG. 6.2 – *Parallèle imagé avec un chantier. Les propositions sont des briques de base et les états des ensembles de briques. L'objectif est de construire la fresque en haut à gauche. Le domaine de planification définit des groupes de briques imagés ici par des pans de murs pré-construits dont quelques exemples sont présentés en haut à droite. Le schéma disjonctif "à la* ***Graphplan****" vise à accéder directement aux briques de bases (en bas à gauche) - c'est-à-dire aux propriétés individuelles. Nous proposons de profiter de la disponibilité offerte de ces propriétés pour reconstruire des groupes adaptés (les nouveaux pans de murs pré-construits en bas à droite), c'est-à-dire des ensembles d'états particuliers et adaptés.*

6.2 Variété de branchements : une étude de cas

Les notions de modélisation, représentation, arborescence et branchement dans l'espace de recherche sont essentielles dans notre démarche. Avant de les formaliser, nous allons les appréhender au travers d'un exemple. Cet exemple est tiré d'un problème plus complexe de poursuite de cible.

6.2.1 Un problème de poursuite de cible

Le problème que nous considérons est étudié en détail dans [20, 19]. Il s'agit d'un problème décisionnel en robotique mobile (cf. figure 6.3).

Deux robots mobiles terrestres sont considérés. L'un est contrôlé par le système décisionnel - c'est *l'observateur*. Le comportement de l'autre est inconnu a priori. Seule sa vitesse maximale de déplacement est donnée. C'est la *cible*. L'objectif de l'observateur est de maintenir un contact visuel avec la cible par l'intermédiaire d'une caméra dont il est doté. C'est une mission de surveillance. Pour coller au plus près des conditions potentielles d'exploitation, l'observateur n'évalue sa position que par des capteurs odométriques et ne peut tirer parti de systèmes externes de localisation absolue qu'en certaines régions de l'espace. L'évaluation de la position de la cible est réalisée par l'observateur par ses propres moyens, et relativement à sa propre position.

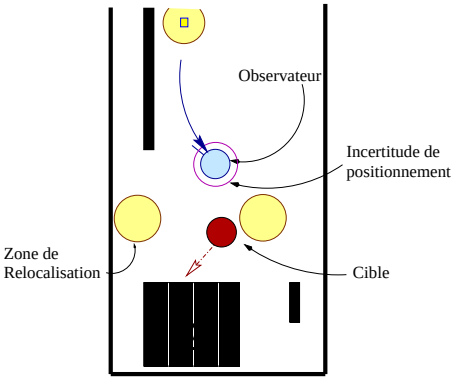


FIG. 6.3 – *Problème de poursuite de cible. A gauche, les robots utilisés pour les expérimentations ([20]).*

Bien-sûr il ne s'agit pas là d'une instance du problème classique de planification. Tout d'abord, les objectifs ne sont pas terminaux. Par ailleurs, l'environnement du système décisionnel est dynamique et incertain :

- dynamique car même quand l'observateur reste inactif la cible peut bouger ;
- incertain car les mouvements de l'observateur sont entâchés d'incertitude, et en conséquence sa position et l'évaluation de celle de la cible le sont aussi.

Malgré cet incrément de complexité, ce problème s'avère très structuré. De nombreuses connaissances spécifiques à cette structure peuvent contribuer à guider la recherche d'une solution. C'est ce potentiel d'exploitation de la struture sous-jacente à l'espace de recherche qui rend ce problème intéressant dans le contexte de nos travaux.

6.2.2 Notre cas d'étude, sa formalisation et la représentation de cette dernière

Notre exemple est issu de ce problème de poursuite. Figeons l'environnement à un instant donné (cf. figure 6.4). Mettons de côté toute incertitude de positionnement et de déplacement. Compte tenu de sa vitesse maximale, sur un horizon temporel donné, les déplacements possibles de la cible forment une zone connue centrée sur sa position actuelle. Notre modeste problème-exemple peut alors se formuler comme suit : maximiser la visibilité de cette zone (en termes de proportion de couverture par le champ de la caméra).

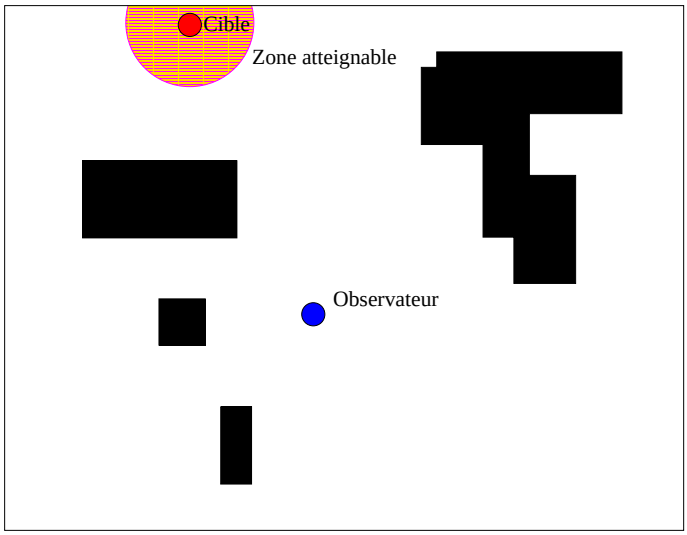


FIG. 6.4 – *L'environnement de l'exemple, avec ses obstacles. La cible est tout à fait en haut, entourée de sa zone atteignable. L'observateur est au centre. Dans cette position il ne voit que partiellement la zone atteignable de la cible.*

Le modèle de l'environnement est simple à exprimer. L'observateur peut se déplacer dans toutes les directions. Les obstacles bloquent à la fois ses déplacements et sa visibilité. La zone visible par l'observateur dépend de sa position relativement aux obstacles et de la portée (fixe) de la caméra. Ce modèle contient une infinité d'états (de par l'infinité de positions possibles) et une infinité de transitions.

Nous choisirons de discrétiser l'espace (cf. figure 6.5), et de ne garder que quatre déplacements élémentaires (`Nord, Sud, Est, Ouest`). Ces restrictions nous ramènent à un nombre fini d'états et de transitions.

Bien-sûr, le mode de discrétisation est totalement arbitraire. Seules les positions retenues seront exploitables.

Plutôt que d'énumérer tous les états possibles, ainsi que les transitions entre eux, nous préférons utiliser une représentation en intension. Nous retiendrons pour chaque état les deux coordonnées de l'observateur ainsi que le pourcentage visible de la zone.

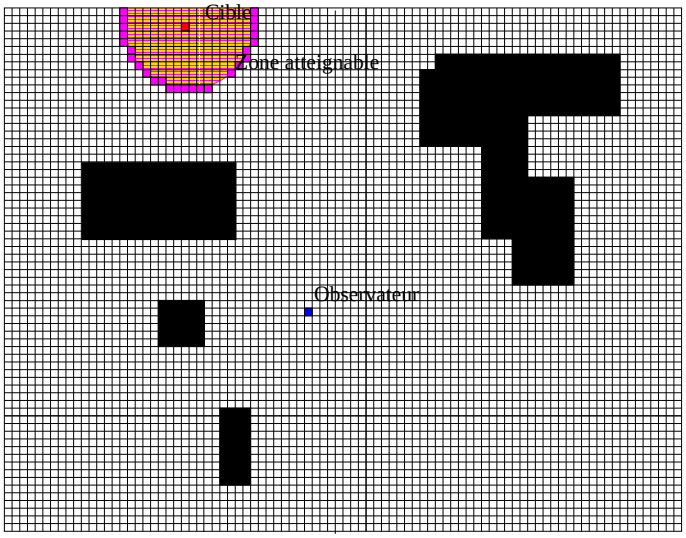


FIG. 6.5 – *L'environnement de l'exemple, une fois discrétisé. Les positions de la cible et de l'observateur sont deux cellules élémentaires.*

6.2.3 Espace de recherche arborescent (arborescence totale)

Comme nous l'avons décrit précédemment, une approche arborescente parcourt l'espace de recherche à partir de la position initiale, en appliquant de manière indépendante chaque déplacement élémentaire. Compte tenu de la forte connectivité de cet espace, les branches se recoupent souvent et de nombreux états sont redécouverts et exploités de nombreuses fois. C'est ce qui explique l'apparence de grille du début de recherche arborescente présenté dans la figure 6.6.

En fait, tous les chemins partant de la position initiale sont construits avec précision. Ceci laisse la possibilité de donner la priorité à l'exploration de tel ou tel chemin en particulier, ce qui revient à donner priorité à certaines branches de l'arbre de recherche.

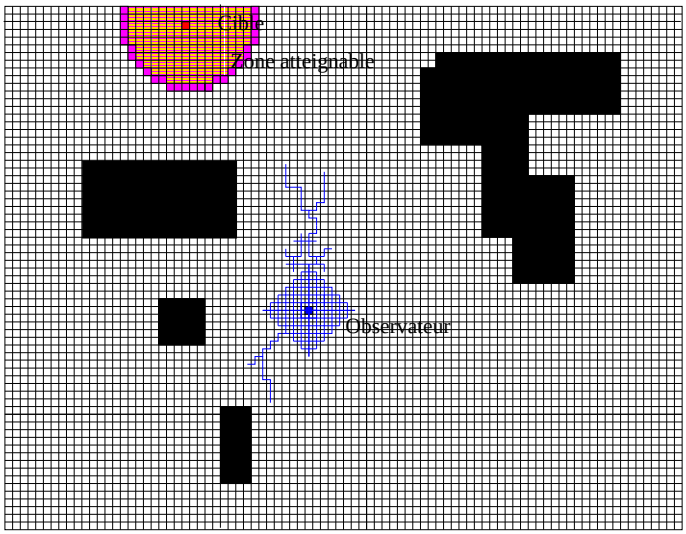


FIG. 6.6 – *Avec une approche arborescente, tous les déplacements élémentaires sont évalués comme autant d'alternatives indépendantes. Pour chaque position déjà considérée, le chemin pour y parvenir est explicitement connu. Quelques branches spécifiques sont représentées, en direction du haut, ou vers le bas à gauche. Ces branchements permettent éventuellement de favoriser certaines orientations de la recherche (ici vers le haut).*

6.2.4 Espace de recherche à la `Graphplan` (arborescence nulle)

Dans une approche disjonctive à la **`Graphplan`**, l'arborescence est nulle. Les chemins pour parvenir aux diverses positions accessibles ne sont plus distingués. Seul compte le fait de savoir que ces positions sont accessibles. A partir de l'ensemble de ce qui est accessible, à une itération donnée de l'étude, il est simple de construire ce qui accessible en un coup de plus. Il suffit d'agrandir un peu la surface déjà obtenue (cf. figure 6.7).

Ici il n'y a pas de problème de duplication de travail. La question de savoir s'il est pertinent ou non de distinguer deux états ne se pose pas. Aucun état n'est distingué en tant qu'individualité. En contrepartie, il n'y a pas d'orientation possible de la recherche. Il n'y a pas de branches. Il est donc impossible de donner priorité à certaines d'entre elles. En revanche, l'étude d'atteignabilité est rapide.

6.2.5 Espace de recherche structuré via une arborescence intermédiaire

Etudions un instant la structure du problème qui est posé.

- Il y a des positions desquelles l'observateur peut voir toute la zone. C'est une de ces positions qui doit être atteinte.
- Certaines positions sont trop éloignées de la zone pour permettre de la voir ou de la voir entièrement. Dans ce cas, il faut se rapprocher.
- La zone peut aussi n'être que partiellement visible, voire totalement invisible, à cause de la présence d'obstacles. Il faut alors changer la position angulaire de l'observateur par rapport à l'obstacle et la cible.

La figure 6.8 a reconstitué cette géométrie du problème.

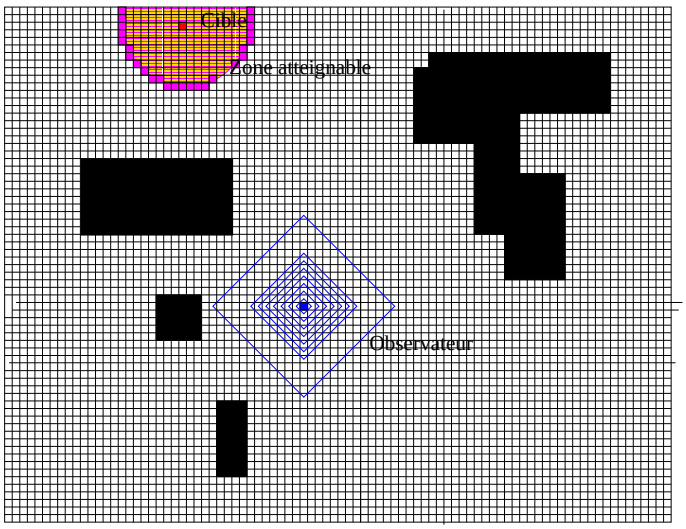


FIG. 6.7 – *Avec une approche disjonctive à la* **Graphplan**, *les positions accessibles sont découvertes de proche en proche et sont considérées comme un tout. Ce comportement fait penser à une onde qui s'éloigne progressivement de la position initiale au fur et à mesure que l'étude d'atteignabilité progresse. Ici, les chemins sont gommés. Seules les positions atteignables subsistent.*

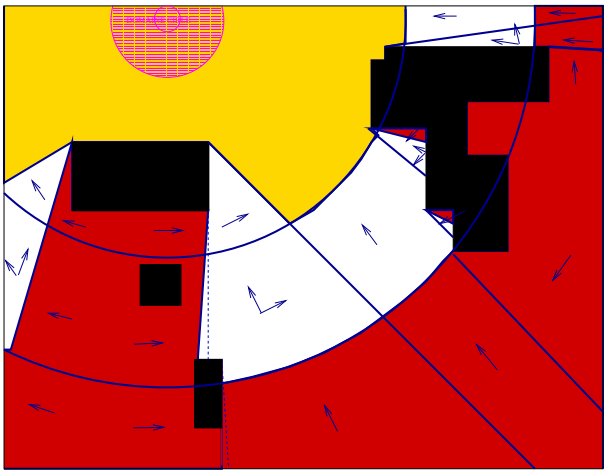


FIG. 6.8 – *Dans ce problème, beaucoup de positions différentes de l'observateur sont équivalentes sur le plan de la visibilité qu'elles offrent sur la cible et de l'action de déplacement qu'il faut y exécuter. Dans les zones en gris sombre, la visibilité de la zone cible est nulle. A l'opposé, dans les zones en gris clair, la visibilité est totale. Dans les zones blanches, la visibilité est partielle - soit à cause de contraintes de distance dues la portée de la caméra, soit à cause de contraintes angulaires dues l'opacitée des obstacles.*

Chacune de ces zones contient plusieurs positions possibles pour l'observateur. Elles y sont toutes à peu près équivalentes. Les considérer une par une avec la précision de l'approche arborescente est inutile. Il n'est pas pertinent d'en faire des cas distincts. Par contre, d'une zone à l'autre, les situations sont effectivement différentes et doivent être distinguées.

C'est cette structure que nous avons repris ici pour morceler l'espace de recherche. Au sein d'une même zone, toutes les positions sont considérées globalement. Pour elles toutes, les mêmes directives globales de déplacement sont utilisées. Par contre, si un changement de zone intervient, il est pertinent de distinguer ce nouveau cas, en lui consacrant une branche spécifique. Au sein de cette branche, les directives globales de déplacement ne sont pas les mêmes. La figure 6.9 représente cette stratégie de morcellement.

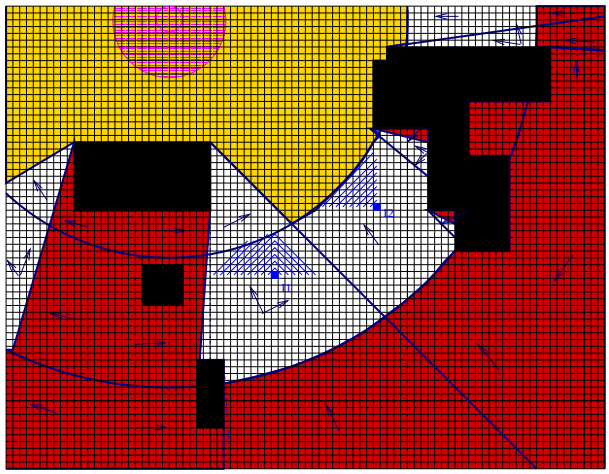


FIG. 6.9 – *La structuration géométrique du problème décisionnel est exploitée par l'arborescence intermédiaire de l'approche présentée ici. Nous retrouvons des sortes de propagations d'ondes - comme en arborescence nulle. Cependant elles sont ici confinées et orientées. Ces contours et ces changements d'orientations correspondent à des branchements dans l'étude d'atteignabilité - comme dans une approche arborescente. Deux états initiaux distincts sont présentés mettant en évidence deux branches de cet arbre de propagation. En effet, les branchements se font ici au niveau d'ensembles d'états - des zones - plutôt qu'au niveau des états individuels.*

6.3 Formalisation

Nous avons présenté notre voie d'investigation de manière intuitive. Nous allons à présent nous faire plus précis.

Le cœur de notre problématique réside dans la définition des objets manipulés par le planificateur et la manière dont celui-ci les manipule, la manière dont il y accède. Le terme "objet" est volontairement vague. Il va être clarifié dans le premier paragraphe de cette section. Celui-ci précise le cadre de travail de la planification. En particulier il y est décrit comment cet ensemble "d'objets" représente le monde considéré par le planificateur.

Ce préalable nous permet dans la suite de montrer comment cette mécanique de représentation et de manipulation est déclinée et exploitée dans les deux approches de planification qui nous intéressent : l'approche arborescente (à la `FSS`) et l'approche disjonctive (à la `GraphPlan`). Les présenter dans cette ordre souligne le processus de la planification disjonctive qui consiste à briser les "objets complexes" dans le but d'avoir accès aux "objets les plus élémentaires". Ce faisant, toute arborescence est détruite, et les avantages qui lui sont liés sont perdus.

Il faut profiter de l'accès désormais possible aux objets élémentaires pour reconstruire des objets plus complexes **adaptés** à la structure du problème à traiter, c'est-à-dire construire un arbre de recherche sur mesure.

6.3.1 Structure d'atteignabilité dans l'espace d'états

Au niveau de la planification, deux éléments essentiels sont à considérer : d'une part le modèle du système étudié et d'autre part la formalisation de ce modèle. Il est important de garder à l'esprit que le modèle est une donnée de la tâche de planification. Le problème de sa correspondance effective au système réel doit se poser en amont.

De la même manière, le potentiel de manipulation de ce modèle est figé lors des choix de sa formalisation. Ce paragraphe a pour objet de poser explicitement le cadre de travail de la planification. Ce faisant, les limitations que nous venons d'évoquer vont se concrétiser.

La réalité peut être modélisée par un ensemble de transitions entre états. Pour le problème classique de planification, toutes les transitions retenues dans le modèle sont déclenchées par des actions de l'agent.

Définition : Modèle à états

Nous définissons le modèle à états du système par :

- un ensemble d'états $\Omega = \{\omega_1,\omega_2,...,\omega_n\}$;
- un ensemble d'actions $A = \{a_1,a_2,...,a_m\}$;
- une fonction définissant les transitions entre états occasionnées par l'exécution des différentes actions. Nous noterons cette fonction T_m car elle définit les transitions retenues dans le modèle :

Définition : T_m, fonction de transition du modèle à états

$T_m : \Omega,A \rightarrow \Omega \cup \{*\}$
$T_m(\omega_1,a) = \omega_2 \in \Omega$ si a est exécutable dans ω_1,
$T_m(\omega_1,a) = *$ sinon.

Par exemple, ω_1 peut être l'état suivant : *Robby* se trouve dans la pièce 1, sa pince gauche est vide, ainsi que sa pince droite ; les balles B_1 et B_2 sont aussi dans la pièce 1 et la balle B_3

est dans la pièce 2 ; les pièces 1 et 2 sont connectées, de même que les pièces 1 et 3 et les pièces 3 et 2. ω_2 peut ne différer de ω_1 que du seul fait que la balle B_2 est dans la pince gauche de *Robby*. Si l'action a_1 est prendre la balle B_2 dans la pince gauche, nous avons $T_m(\omega_1,a_1) = \omega_2$.

T_m définit une relation binaire sur Ω. Notons-la R_m.

Définition : R_m, relation de transition du modèle à états (cf. Figure 6.10)

$$(\omega_1,\omega_2) \in R_m \; ssi \; \exists a \in A \,/\, T_m(\omega_1,a) = \omega_2$$

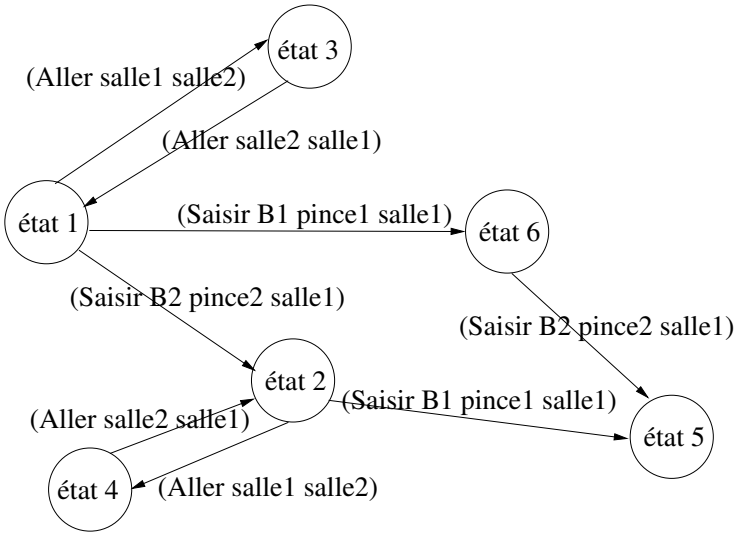


FIG. 6.10 – *R_m est une relation qui transcrit les états et les transitions entre ces états qui ont été retenus dans le modèle du système considéré. Voici ici un extrait de la relation correspondant au domaine de Robby - que nous utilisons dans la plupart de nos exemples. Il est important de noter que rien ne permet de savoir si l'état 2 diffère peu ou beaucoup de l'état 1 en termes des propriétés de l'état.*

Ce qui est donné au planificateur est une représentation de ce modèle. Considérons la représentation en intension largement utilisée en planification. Elle est fondée sur un ensemble P de propriétés concernant les états ; $P = \{p_1,p_2,...,p_k\}$.

Nous noterons :

- $\mathbf{P}_{\omega_i}$ l'ensemble des propriétés vraies pour ω_i ;
- $\mathbf{\Omega_{p_i}}$ l'ensemble des états pour lesquels p_i est vraie.

La fonction de transition est formulée en termes de cette représentation :

Définition : T_r, fonction de transition de la représentation en propriétés

$T_r : 2^P,A \rightarrow 2^P \cup \{*\}$

$T_r(\{p_1,p_2,...,p_t\},a) = \{p_h,...,p_j\} \in 2^P$ si a est exécutable dans les états de

$$\bigcap_{i \in \{1,2,...,t\}} \Omega_{p_i}$$

$T_r(\{p_1,p_2,...,p_t\},a) = *$ sinon.

Cette fonction de transition s'éloigne de la notion d'état. Elle ne considère que les propriétés qui font qu'une action est déclenchable ou non. Considérant un ensemble de propriétés, les actions dont elles sont les uniques préconditions pourront être déclenchées dans tous les états pour lesquels ces propriétés sont vraies. A chaque fois, déclencher cette action entraînera l'obtention du même ensemble de propriétés effets.

Il est possible que la structure de P corresponde exactement à celle d'Ω, c'est-à-dire qu'il y ait autant de propriétés que d'états et que chaque Ω_p contienne un seul état (cf. figure 6.11). Il est aussi possible que chaque état soit décrit par plusieurs propriétés (cf. figure 6.12) . De manière duale, chaque propriété permet alors de discriminer des ensembles d'états plutôt que des états individuels. Les représentations en intension le plus souvent utilisées en planification correspondent au second cas.

T_r définit une relation binaire sur 2^P.

Notons-la R_r. R_r est la représentation de R_m qui est fournie au planificateur. Elle correspond au domaine de planification.

Définition : R_r, relation de transition de la représentation en propriétés
$(\{p_1,p_2,...,p_t\},\{p_h,...,p_j\}) \in R_r \; ssi \; \exists a \in A \,/\, T_r(\{p_1,p_2,...,p_t\},a) = \{p_h,...,p_j\}$

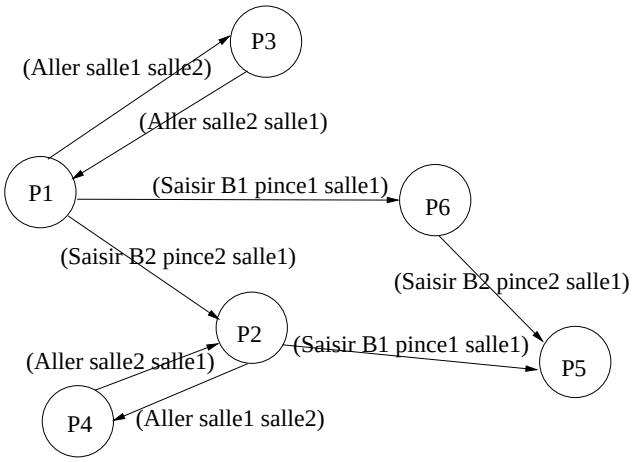


FIG. 6.11 – *Sur cette figure, la représentation colle exactement au modèle. Chaque propriété correspond à un état.*

Propriétés : Il est attendu d'une représentation "fidèle" du modèle considéré qu'elle remplisse les trois propriétés suivantes :

1. La représentation distingue tous les états du modèle :
   $\forall(\omega_i,\omega_j) \in \Omega^2, \; \omega_i \neq \omega_j \Leftrightarrow P_{\omega_i} \neq P_{\omega_j}$
2. La représentation transcrit T_m :
   $\forall \omega \in \Omega \; \forall a \in A$
   $T_r(P_\omega,a) = P_{T_m(\omega,a)} \; si \; T_m(\omega,a) \neq \; *$
   $T_r(P_\omega,a) = \; * \; sinon$
3. La représentation transcrit R_m :
   $\forall(\omega_i,\omega_j) \in \Omega^2, \; (\omega_i,\omega_j) \in R_m \Leftrightarrow (P_{\omega_i},P_{\omega_j}) \in R_r$

Pour chaque problème, un planificateur considère une relation de transition adaptée censée lui permettre de trouver plus facilement une solution. C'est cette relation spécifique qui constitue son espace de recherche. Celui-ci doit exploiter $R-r$ et T_r pour mettre en relief les relations d'atteignabilité qui peuvent exister entre l'état initial et l'état final du problème considéré.

Les paragraphes suivant présentent les relations spécifiques à `FSS` et `Graphplan` pour ensuite décrire la notion de morcellement que nous proposons pour mieux contrôler la forme de ces relations d'atteignabilité.

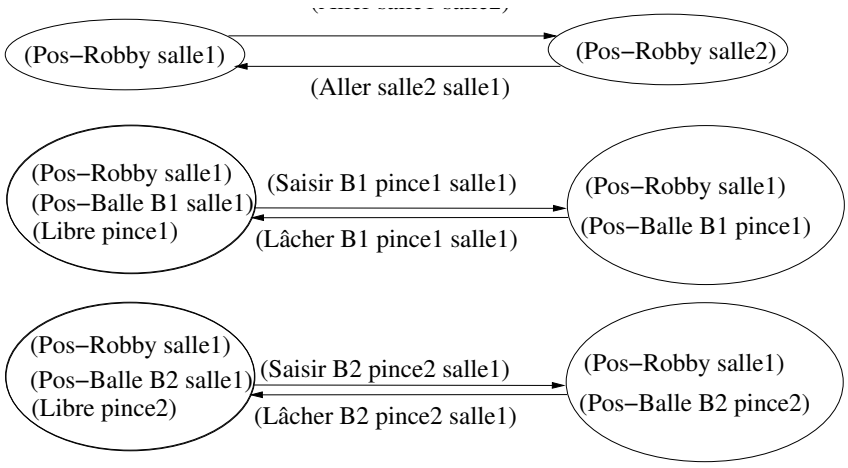


FIG. 6.12 – *Cette figure dcrit une partie d' une représentation en intension dans laquelle un état est décrit par un ensemble de propriétés. Par exemple, la transition entre les états 1 et 2 dans la figure 6.10 ne concerne en fait que la position de Robby, celle de la balle B2, et l'état de la pince 2 de Robby. La relation représentée ici, montre que cette transition est déclenchable dans tous les états qui rassemblent ces propriétés.*

6.3.2 Structuration arborescente de l'espace de recherche à la FSS

Dans le cadre d'une approche de planification arborescente, l'espace de recherche correspond à la relation d'atteignabilité précise et explicite de tous les états à partir de l'état initial du problème considéré. Construire cet espace de recherche revient à "dérouler" R_m jusqu'à en éliminer tous les circuits. Ainsi, même si R_r est bâtie sur une représentation en intension où plusieurs propriétés sont généralement nécessaires à la description d'un état, l'espace de recherche considèrera uniquement des paquets de propriétés qui correspondent à des états individuels.

Une conséquence inévitable de l'élimination des circuits est la possible multiplication des instances de chacun des états atteints. En effet, la présence de circuits dans R_m signifie que certains états peuvent être atteints de plusieurs façons. Chacune de ces instances est explicitée individuellement dans l'espace de recherche arborescent. Bien entendu, elles aboutissent toutes au même état. Il y a donc autant d'instances de cet état qu'il y a de manières de l'atteindre.

Pour comprendre, une analogie avec la représentation du réseau ferré français peut être utilisée. Le réseau peut être dessiné sur une carte de France. Des livrets sont aussi édités indiquant pour chaque train la liste des gares desservies. Une ville comme Lyon apparaît dans de nombreux livrets, car de nombreux trains y passent. Pourtant, Lyon n'apparaît qu'une seule fois sur la carte du réseau ferré.

Concrètement, notons R_a la relation correspondant à la structure arborescente de l'espace de recherche. R_a porte sur un ensemble de nœuds $N = \{n_1,n_2,...,n_m\}$ tel que chaque nœud est associé à un seul état $\omega \in \Omega$. Bien évidemment, plusieurs nœuds différents peuvent être associés à un même état. Notons *etat* la fonction retournant l'état associé à un nœud donné :
$etat : N \rightarrow \Omega$
$etat(n) = \omega$

R_a peut alors se définir de la manière inductive suivante.

Définition : R_a, relation de transition de la représentation en propriétés d'un espace de recherche à structure arborescente

Nous définissons la distance d_a :
$d_a \ : \ N^2 \rightarrow \mathbb{N}$
$\forall (n_i,n_j) \in \mathbb{N}^2$, $d_a(n_i,n_j) = l \in \mathbb{N}$ *ssi* $\exists \{n_m,n_{m+1},\ldots,n_{m+l}\}$ *tels que* $n_m = n_i$, $n_{m+l} = n_j$, *et* $\forall k \in \{m+1,\ldots,m+l\}$ $(n_{k-1},n_k) \in R_a$

Schéma inductif de R_a

Soit $n_0 \in N$ le noeud associé à l'état initial du problème ω_{init}.
Nous notons N_a^k l'ensemble des nœuds n_i tels que $d_a(n_0,n_j) = k$.

- Initiation :
  $\forall \omega \in \Omega / (\omega_{init},\omega) \in R_m$, $\exists n \in N$ / $etat(n) = \omega$ *et* $(n_0,n) \in R_a$
- Récurrence
  Soit $l \in \mathbb{N}$ $\forall n_l \in N_a^l$ $\forall n_k \in N$, $(n_l,n_k) \in R_a$ *ssi* $(etat(n_l),etat(n_k)) \in R_m$

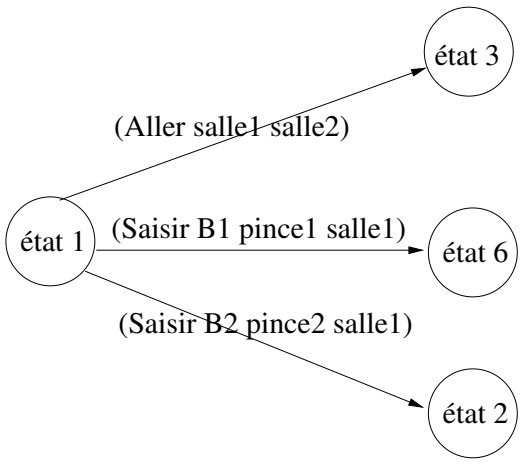


FIG. 6.13 – *En supposant que l'état initial du problème traité soit etat*1, *voici l'initiation de l'arbre de recherche (d'après R_m représenté dans la figure 6.10).*

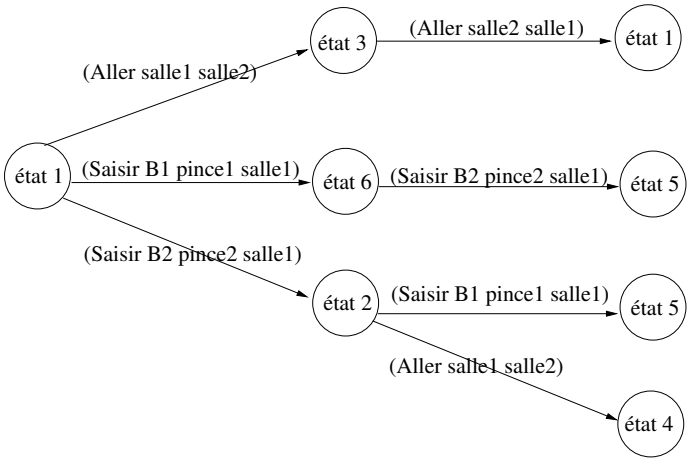


FIG. 6.14 – *Voici une plus grande portion de l'arbre de recherche A_{R_a}. Notons que les états* 1 *et* 5 *sont présents plusieurs fois (duplication dans A_{R_a}) à cause d'un circuit dans R_m.*

6.3.3 Structuration disjonctive de l'espace de recherche à la `Graphplan`

Les approches disjonctives s'attachent aux propriétés de la représentation plutôt qu'aux états. De fait, en manipulant directement les propriétés, ce sont des ensembles d'états qui sont manipulés et non des états individuels. Ainsi, alors que les approches arborescentes s'intéressent à l'atteignabilité des états, donc à l'atteignabilité des conjonctions de propriétés qui décrivent les états individuels, les approches disjonctives s'intéressent à l'atteignabilité des propriétés individuelles - lesquelles représentent des disjonctions d'états.

Par ailleurs, une approche "à la Graphplan" s'attachant à l'atteignabilité des propriétés indépendemment les unes des autres, manipule aussi des disjonctions de propriétés. En effet, quand l'espace de recherche indique que plusieurs propriétés sont atteignables en deux coups, il n'indique pas qu'elles le sont en conjonction. C'est d'ailleurs pour cela que `Graphplan` calcule par ailleurs des relations de mutuelle exclusion.

Ainsi, les approches disjonctives "à la Graphplan" construisent leurs espaces de recherche en déroulant R_r à partir du groupe de propriétés présentes dans l'état initial, sans tenir compte des regroupements de propriétés qui sont définis dans R_r.

Notons R_d la relation de transition d'une structuration disjonctive de l'espace de recherche. R_d porte sur un ensemble de nœuds $N = \{n_1,n_2,...,n_m\}$. Cette fois chaque nœud est associé à un sous-ensemble de P.

Notons $prop$ la fonction transcrivant cette association entre nœuds et ensembles de propriétés.
$prop : N \rightarrow 2^P$
$prop(n) = \{p_1,p_2,...,p_m\}$.

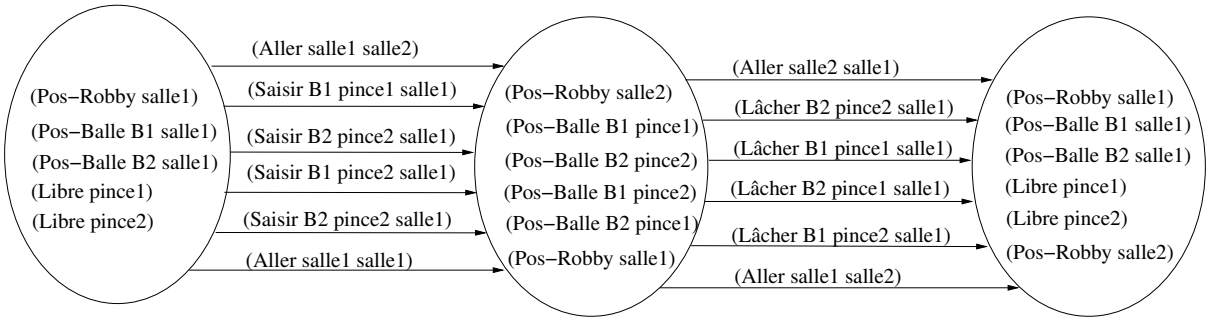


FIG. 6.15 – *La relation R_d est une séquence. Chaque nœud est un ensemble non structuré de propriétés. Notons que certaines propositions apparaissent plusieurs fois, mais jamais au sein du même niveau.*

R_d peut alors se définir de la manière inductive suivante.

Définition : R_d, relation de transition de la représentation en propriétés d'un espace de recherche à structure disjonctive

Nous définissons la distance d_d :

$d_d : N^2 \rightarrow \mathbb{N}$

$\forall (n_i,n_j) \in \mathbb{N}^2$, $d_d(n_i,n_j) = l \in \mathbb{N}$ *ssi* $\exists \{n_m,n_{m+1},\ldots,n_{m+l}\}$ *tels que* $n_m = n_i$, $n_{m+l} = n_j$, *et* $\forall k \in \{m+1,\ldots,m+l\}$ $(n_{k-1},n_k) \in R_d$

Schéma inductif de R_d

Soit $n_0 \in N$ le noeud associé aux propriétés de l'état initial du problème $P_{\omega_{init}}$.

Nous notons N_d^k l'ensemble des noeuds n_i tels que $d_d(n_0,n_j) = k$.

- Initiation :

  Soit $n_1 \in N$

  $\forall P_i \subset prop(n_0)$, $\forall P_j \subset P$, $(P_i,P_j) \in R_r \Leftrightarrow Pj \subset prop(n1)$

  $(n_0,n_1) \in R_d$

- Récurrence

  Soit $l \in \mathbb{N}$ $N_d^l = \{n_l\}$(il s'agit d'un singleton)

  Alors $N_d^{l+1} = \{n_{l+1}\}$ *tel que*

$$prop(n_{l+1}) = \bigcup_{P_j \subset P \,/\, \exists P_i \subset prop(n_l)(P_i,P_j) \in R_r} P_j$$

  $(n_l,n_{l+1}) \in R_d$

Un nœud est généralement associé à plusieurs éléments de P. Tous les ensembles d'états pouvant être formés par composition de ces éléments sont alors considérés. C'est la base de la planification disjonctive : chaque nœud représente une disjonction de possibilités.

Au sein de chaque niveau d'atteignabilité, chaque propriété n'est représentée qu'une seule fois pour toutes les conjonctions de propriétés dans lesquelles elle pourrait être atteinte. Plusieurs instances d'une même propriété ne peuvent apparaître que dans plusieurs niveaux différents.

En contre-partie, les états atteignables à chaque niveau ne sont pas clairement définis. En effet, en éliminant les groupements de propriétés, la planification disjonctive "à la Graphplan"

perd toute information sur les conjonctions de propriétés légitimes. Au sein d'un niveau, toutes les propriétés ne sont évidemment pas atteignables en même temps.

6.3.4 Introduire de l'arborescence dans une structuration disjonctive de l'espace de recherche

Notre objectif est de réintroduire de l'arborescence dans la planification disjonctive. Cela implique de reformer des groupes de propriétés appropriés. Chaque nœud de la séquence est ainsi morcelé en plusieurs nœuds. Pour cela, il suffit de répartir les propriétés associées à un nœud en plusieurs *classes*. Pour chaque classe utilisée, un nouveau nœud est produit de sorte que chaque nouveau noeud est associé à une classe unique de propriétés.

Notation : Ensemble de classes

Nous notons $\mathcal{C}$ l'ensemble des *classes* de propriétés.
$\forall C_i \in \mathcal{C}, C_i \in 2^P$
$\forall P_i \subset P \; \forall C_j \in \mathcal{C}$, nous notons $C_j^{P_i}$ l'ensemble des propriétés de P_i dans C_j :
$C_j^{P_i} = P_i \cap C_j$

Afin de définir le morcellement de l'ensemble de propriétés d'un nœud en classes, nous introduisons la fonction de morcellement $\mathcal{F}_{morc}$.

Définition : *fonction de morcellement* $\mathcal{F}_{morc}$

Nous appellerons *fonction de morcellement* une fonction qui associe à tout ensemble P_j de propriétés pris dans une classe donnée un ensemble de classes permettant de définir un recouvrement de P_j :
$\mathcal{F}_{morc} \; : \; 2^P \mathrm{x} \mathcal{C} \longrightarrow 2^{\mathcal{C}}$
$\forall P_j \subset P, \forall C_j \in \mathcal{C}$ tels que $P_j \subset C_j$
$\mathcal{F}_{morc}(P_j,C_j) \; = \; \{C_1,C_2,...,C_n\} \; / \; \bigcup_{C_i \in \mathcal{F}_{morc}(P_j,C_j)} C_i^{P_j} = P_j$

Chaque C_i correspond à une classe particulière de propriétés. A chacune de ces classes est associée un nœud n_i dans la structuration morcelée R_{morc} de l'espace de recherche.

Pour chaque sous ensemble de $prop(n)$ recensé suivant $\mathcal{F}_{morc}(prop(n),C)$, il est alors possible d'introduire une branche distincte (cf. figure 6.16).

Nous pouvons ainsi définir de manière inductive une relation de transition spécifique - que nous notons R_{morc}.

Définition : R_{morc}, relation de transition de la représentation en propriétés d'un espace de recherche morcelé selon $\mathcal{F}_{morc}$

Nous définissons la distance d_{morc} :
$d_{morc} \; : \; N^2 \rightarrow \mathbb{N}$
$\forall (n_i,n_j) \in \mathbb{N}^2$, $d_{morc}(n_i,n_j) = l \in \mathbb{N}$ *ssi* $\exists \{n_m,n_{m+1},\ldots,n_{m+l}\}$ *tels que* $n_m = n_i$, $n_{m+l} = n_j$, *et* $\forall k \in \{m+1,\ldots,m+l\}$ $(n_{k-1},n_k) \in R_{morc}$

Schéma inductif de R_{morc}

Soit $n_0 \in N$ le nœud associé aux propriétés de l'état initial du problème $P_{\omega_{init}}$.
Nous notons N_{morc}^k l'ensemble des nœuds n_i tels que $d_{morc}(n_0,n_j) = k$.

- Initiation :
  Soit $n_1 \in N$ *tel que* $(n_0,n_1) \in R_d$ (relation d'accessibilité de la structuration disjonctive de l'espace de recherche).

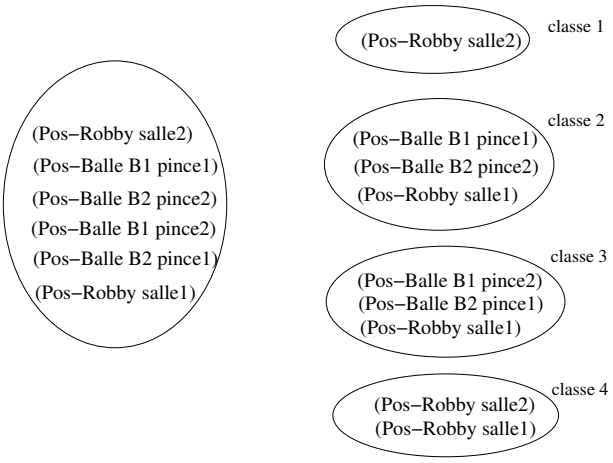


FIG. 6.16 – *A gauche un niveau de R_d. A droite un morcellement possible. Notons que certaines propriétés appartiennent à plusieurs classes. Notons aussi que l'ensemble des classes couvre l'intégralité de l'ensemble original de propriétés. Chaque nœud obtenu dans le morcellement peut devenir la racine d'une branche distincte.*

Soit $C_1 \in \mathcal{C}$ la classe associée à $prop(n_1)$. $\forall C_i \in \mathcal{F}_{morc}(n_1,C_1)$, soit n_i / $prop(n_i) = C_i^{prop(n_1)}, (n_0,n_i) \in R_{morc}$
Ainsi, pour toute classe C_i de $\mathcal{F}_{morc}(n_1,C_1)$, on crée un nœud n_i dans la structuration morcelée de l'espace de recherche. Ce nœud est associé aux propositions de n_1 correspondant à la classe considérée. (n_0,n_i) marque la relation d'atteignabilité entre n_0 et n_i, morcelée à partir de la relation d'accessibilité disjonctive (n_0,n_1).

– Récurrence
Soit $l \in \mathbb{N}$
$\forall n_l \in N^l_{morc}$, soient C_l la classe associée à n_l, et P_{l+1} l'ensemble des propriétés atteignables à partir de n_l,

$$P_{l+1} = \bigcup_{P_j \subset P \,/\, \exists P_i \subset prop(n_l)(P_i,P_j) \in R_r} P_j$$

$\forall C_i \in \mathcal{F}_{morc}(P_{l+1},C_l)$, soit n^i_{l+1} / $prop(n^i_{l+1}) = C_i^{P_{l+1}}, (n_l,n^i_{l+1}) \in R_{morc}$

Il est notable que les ensembles N^l_{morc} ne sont pas des singletons, contrairement aux ensembles de nœuds équidistants de la relation d'atteignabilité de l'espace de recherche à structure disjonctive. Ainsi, à partir du morcellement des niveaux de propriétés en sous-ensembles, suivant leurs classes d'appartenance, nous obtenons effectivement une structuration arborescente.

De manière duale, nous pouvons construire une arborescence de classes à partir des actions du domaine. Nous obtenons au final un morcellement des ensembles de propriétés. Cette modélisation duale est présentée ci-après.

Nous introduisons une fonction de branchement $\mathcal{F}_{branch}$.

Définition : *fonction de branchement* $\mathcal{F}_{branch}$

Nous appellerons *fonction de branchement* une fonction qui indique pour toute classe C et toute action a la classe à laquelle aboutit l'exécution de a dans C :

$$\mathcal{F}_{branch} \ : \ \mathcal{C}\mathrm{x}\mathcal{A} \longrightarrow 2^{\mathcal{C}}$$

$$\forall C_i \in \mathcal{C}, \forall a \in \mathcal{A}, \ \mathcal{F}_{branch}(C_i,a) \ = \ \{C_1, C_2, \ \ldots \ , C_n\}$$

$$tel\ que \bigcup_{P_k \subset C_i \ / \ T_r(P_k,a) \neq *} T_r(P_k,a) \ \subset \bigcup_{j \in \{1, \ldots, n\}} C_j$$

Ces transitions de classes permettent de répartir les propriétés obtenues par l'exécution des actions correspondantes. Nous pouvons ainsi à nouveau définir de manière inductive une relation de transition spécifique - que nous notons R_{branch}.

Définition : R_{branch}, relation de transition de la représentation en propriétés d'un espace de recherche structuré selon $\mathcal{F}_{branch}$

Nous définissons la distance d_{branch} :

$d_{branch} \ : \ N^2 \rightarrow \mathbb{N}$

$\forall (n_i,n_j) \in \mathbb{N}^2$, $d_{branch}(n_i,n_j) = l \in \mathbb{N}$ *ssi* $\exists \{n_m, n_{m+1}, \ldots, n_{m+l}\}$ *tels que* $n_m = n_i$, $n_{m+l} = n_j$, *et* $\forall k \in \{m+1, \ldots, m+l\}$ $(n_{k-1}, n_k) \in R_{branch}$

Schéma inductif de R_{branch}

Soit $n_0 \in N$ le noeud associé aux propriétés de l'état initial du problème $P_{\omega_{init}}$.

Nous notons N^k_{branch} l'ensemble des noeuds n_i tels que $d_{branch}(n_0,n_j) = k$.

- Initiation :

  Soit $C_0 \in \mathcal{C}$ la classe associée à $prop(n_0)$.

  $$\exists C_{next} \subset 2^{\mathcal{C}} \ tel\ que\ C_{next} = \bigcup_{a \in \mathcal{A}} \mathcal{F}_{branch}(C_0,a)$$

  $C_{next} = \{C_1, C_2, \ \ldots \ , C_m\}$ contient toutes les classes accessibles depuis C_0.

  Pour chacune d'entre elles est créé un nœud. Soit N_{next} l'ensemble de ces nœuds : $N_{next} = \{n_1, n_2, \ \ldots \ , n_m\}$.

  On a alors :

  $$\forall n_i \in N_{next}, prop(n_i) = \bigcup_{a \in \mathcal{A} \ / \ T_r(prop(n_0),a) \neq *} C_i^{T_r(prop(n_0),a)}$$

  $$\forall n_i \in N_{next}, (n_0,n_i) \in R_{branch}$$

- Récurrence

  Soit $l \in \mathbb{N}$

  $\forall n_l \in N^l_{branch}$, soient C_l la classe associée à n_l,

  $$\exists C_{next} \subset 2^{\mathcal{C}} \ tel\ que\ C_{next} = \bigcup_{a \in \mathcal{A}} \mathcal{F}_{branch}(C_l,a)$$

$C_{next} = \{C_1, C_2, \ldots, C_m\}$ contient toutes les classes accessibles depuis C_l.
Pour chacune d'entre elles est créé un nœud. Soit N_{next} l'ensemble de ces nœuds : $N_{next} = \{n_1, n_2, \ldots, n_m\}$.

On a alors :

$$\forall n_i \in N_{next}, prop(n_i) = \bigcup_{P_j \subset prop(n_l)\ a \in \mathcal{A} \,/\, T_r(P_j),a) \neq *} C_i^{T_r(P_j),a)}$$

$$\forall n_i \in N_{next}, (n_l, n_i) \in R_{branch}$$

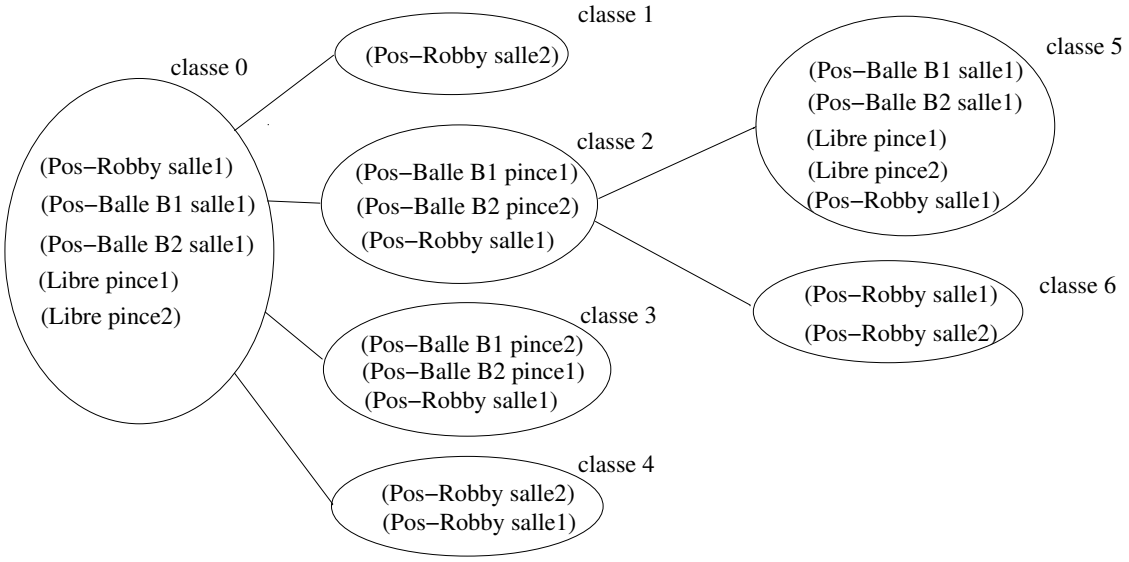


FIG. 6.17 – *R_{morc} peut être arborescente (comme R_a - relation d'accessibilité de la structuration arborescente dans l'espace d'états). Cependant, chaque nœud ne correspond pas nécessairement à un seul état (comme R_d - relation d'accessibilité de la structuration disjonctive de l'espace de recherche).*

Ces deux points de vue sont complémentaires. Ils sont plus ou moins adaptés suivant les motivations pour introduire plus d'arborescence et les structures du domaine que nous voulons exploiter.

En outre, $\mathcal{F}_{morc}$ et $\mathcal{F}_{branch}$ ont la même expressivité. Ainsi, $\mathcal{F}_{morc}$ et $\mathcal{F}_{branch}$ permettent d'obtenir toutes deux les mêmes morcellements de l'espace de recherche.

Propriété : $\forall\,(P_i,P_j) \in 2^P \text{x} 2^P$, $soit\,(n_i,n_j) \in N^2$ / $prop(n_i) = P_i$ *et* $prop(n_j) = P_j$ *alors* :
$\exists \mathcal{F}_{morc}$ / $(n_i,n_j) \in R_{morc} \Leftrightarrow \exists\,\mathcal{F}_{branch}$ / $(n_i,n_j) \in R_{branch}$

Schéma de preuve :

Implication :

Nous utilisons la construction suivante :

$$\forall a \in \mathcal{A}\ \forall C_j \in \mathcal{C}\ \mathcal{F}_{branch}(C_j,a) = \bigcup_{P_k \subset C_j\ /\ Tr(P_k,a) \neq *} \mathcal{F}_{morc}(P_k,C_j)$$

Réciproque :

Nous utilisons la construction suivante :

$$\forall a \in \mathcal{A}\ \forall C_j \in \mathcal{C}\ \mathcal{F}_{morc}(C_j,a) = \bigcup_{a \in \mathcal{A}} \mathcal{F}_{branch}(C_j,a)$$

6.3.5 Propriétés et expressivité d'une structuration à "arborescence intermédiaire

Il existe une multitude de possibilités pour définir $\mathcal{F}_{morc}$ ou $\mathcal{F}_{branch}$.

Le schéma disjonctif "à la Graphplan" est obtenu avec la forme suivante de $\mathcal{F}_{morc}$: $\forall C_j \in \mathcal{C}$, $\forall P_j \in 2^P$, $\mathcal{F}_{morc}(P_j,C_j) = \{C_j\}$. Nous retrouvons alors la structure de l'espace de recherche de la planification disjonctive (R_d).

Propriété : $\exists \mathcal{F}_{morc}$ *telle que* $R_{morc} = R_d$

Preuve :

Nous définissons $\mathcal{F}_{morc}$ de la manière suivante (def 1): $\mathcal{F}_{morc} : 2^P \text{x} \mathcal{C} \longrightarrow 2^{\mathcal{C}}$
$\forall P_j \subset P, \forall C_j \in \mathcal{C}$ tels que $P_j \subset C_j$
$\mathcal{F}_{morc}(P_j,C_j) = \{C_j\}$
Nous allons montrer par induction qu'avec cette définition $R_{morc} = R_d$.

- Initiation :
  D'après la définition de R_{morc}, nous avons :
  Soit $n_1 \in N$ *tel que* $(n_0,n_1) \in R_d$
  Soit $C_1 \in \mathcal{C}$ la classe associée à $prop(n_1)$.
  $\forall C_i \in \mathcal{F}_{morc}(n_1,C_1)$, soit n_i / $prop(n_i) = C_i^{prop(n_1)}$, $(n_0,n_i) \in R_{morc}$
  Or, d'après (def 1), $\mathcal{F}_{morc}(n_1,C_1) = C_1$.
  Donc, $prop(n_i) = prop(n_1)$.
  Nous avons bien $(n_0,n_1) \in R_d$ *et* $(n_0,n_1) \in R_{morc}$.
- Récurrence :
  Soit $l \in \mathbb{N}$
  soit $N_l \subset N$ / $N_l = \bigcup_{k \in \{0,\ldots,l\}} N^k_{morc}$
  Supposons que $\forall (n_i,n_j) \in N_l \text{x} N_l$, $(n_i,n_j) \in R_{morc} \Leftrightarrow (n_i,n_j) \in R_d$.
  Nous allons montrer que cela reste vrai pour N_{l+1}.
  Par hypothèse de récurrence, $N^l_{morc} = N^l_d$.

La définition de R_{morc} nous donne :
$\forall n_k \in N^l_{morc}$, soient C_k la classe associée à n_k, et P_{k+1} l'ensemble des propriétés atteignables à partir de n_k,

$$P_{k+1} = \bigcup_{P_j \subset P \,/\, \exists P_i \subset prop(n_k)(P_i,P_j) \in R_r} P_j$$

$\forall C_i \in \mathcal{F}_{morc}(P_{k+1},C_k)$, soit n^i_{k+1} / $prop(n^i_{k+1}) = C_i^{P_{k+1}}, (n_k,n^i_{k+1}) \in R_{morc}$
Or, ici, $\mathcal{F}_{morc}(P_{k+1},C_k) = \{C_k\}$;
et d'après la définition de R_d, nous savons que N^l_{morc} est un singleton : $N^l_{morc} = \{n_l\}$.
Donc N^{l+1}_{morc} est un singleton : $N^{l+1}_{morc} = \{n_{l+1}\}$ *tel que*

$$prop(n_{l+1}) = \bigcup_{P_j \subset P \,/\, \exists P_i \subset prop(n_l)(P_i,P_j) \in R_r} P_j$$

Ainsi, d'après la définitin de R_d, $N^{l+1}_{morc} = N^{l+1}_d$
$(n_l,n_{l+1}) \in R_d$ et $(n_l,n_{l+1}) \in R_{morc}$

■

Par ailleurs, si $\mathcal{F}_{morc}$ ne génère que des classes de propriétés définissant des états, alors nous retrouvons R_a, c'est-à-dire la relation d'accessibilité d'un graphe de planification arborescent dans l'espace des états. La figure 6.18 le montre sur un exemple.

Propriété : $\exists \mathcal{F}_{branch}$ *telle que* $R_{branch} = R_a$
Preuve :
Nous définissons $\mathcal{C}$ *et* $\mathcal{F}_{branch}$ de la manière suivante (def1):
$\mathcal{C} = \{C_1, \ldots, C_n \,/\, \forall \omega \in \Omega, \exists i \in \{1, \ldots, n\}$ *tel que* $C_i = P_\omega\}$
$\mathcal{F}_{branch} : \mathcal{C}\mathrm{x}\mathcal{A} \longrightarrow 2^{\mathcal{C}}$
$\forall C_i \in \mathcal{C}, \forall a \in \mathcal{A}$ *tels que* $T_m(\Omega_{C_i},a) \neq *$, $\mathcal{F}_{branch}(C_i,a) = \{C_j\}$ *tel que* $\Omega_{C_j} = T_m(\Omega_{C_i},a)$

Nous allons montrer par induction qu'avec cette définition $R_{branch} = R_a$.
Nous supposons que la représentation utilisée est fidèle au modèle considéré, au sens défini dans la section 6.3.1- cf. p.96).

- Initiation :
  Reprenons le schéma inductif de R_{branch}.
  Soit $C_0 \in \mathcal{C}$ la classe associée à $prop(n_0)$.
  Ici, $C_0 = P_{\omega_{init}}$.

$$\exists C_{next} \subset 2^{\mathcal{C}} \text{ tel que } C_{next} = \bigcup_{a \in \mathcal{A}} \mathcal{F}_{branch}(C_0,a)$$

  $C_{next} = \{C_1, C_2, \ldots, C_m\}$ contient toutes les classes accessibles depuis C_0. Elles correspondent à tous les états atteignables en un coup depuis $\omega init$ (d'après (def1) - au début de cette preuve).

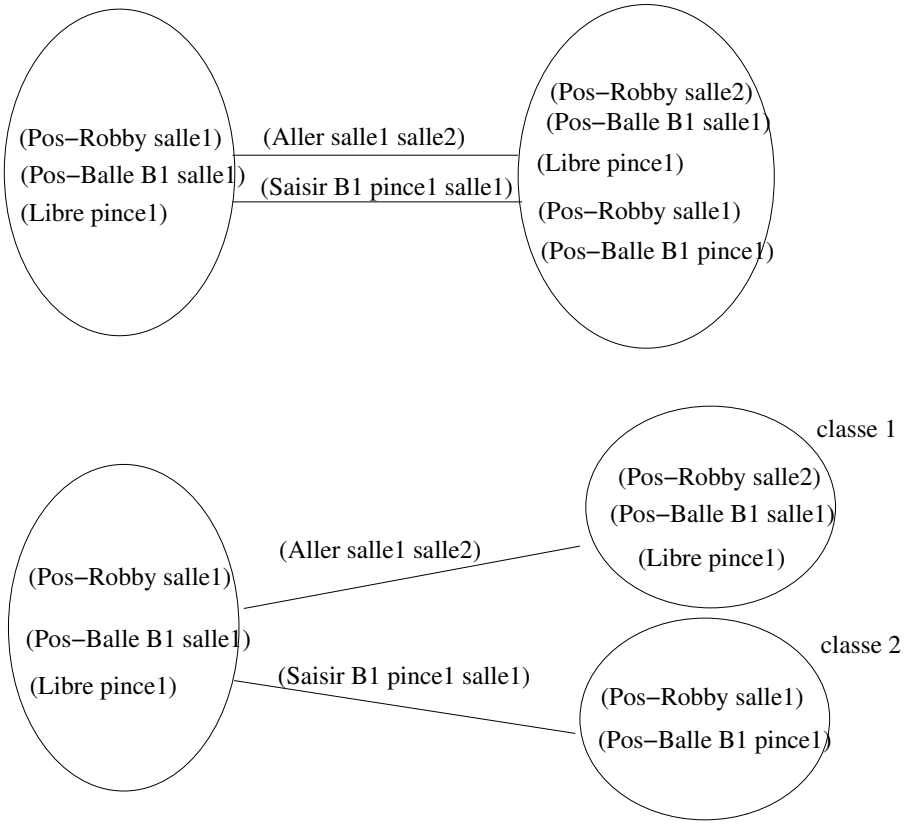


FIG. 6.18 – *Le schéma du haut présente le premier niveau de propositions généré en morcellement nul à partir d'un état initial dans lequel le robot Robby a une seule pince, une seule balle est considérée et deux salles. Le schéma du bas présente un morcellement possible en deux classes pour lequel chaque classe correspond à un état. C'est le début de l'arbre généré par une approche de type* FSS - *approche arborescente dans l'espace d'états.*

$C_{next} = \{P_{\omega_1}, \ \ldots \ , P_{\omega_m}\}$ tel que $\forall i \in \{1 \ \ldots \ m\} \ \exists a_i \in \mathcal{A} \ / \ T_m(\Omega_{init}, a_i) = \omega_i$

Soit N_{next} l'ensemble des nœuds associés à ces classes : $N_{next} = \{n_1, n_2, \ \ldots \ , n_m\}$.
On a alors :

$$\forall n_i \in N_{next}, prop(n_i) = \bigcup_{a \in \mathcal{A} \ / \ T_r(prop(n_0),a) \neq *} C_i^{T_r(prop(n_0),a)}$$

Or, $\forall a \in \mathcal{A}$, $T_r(prop(n_0),a) = P_{T_m(\Omega_{init}, a_i)}$ *si* $T_m(\omega_{init}, a) \neq *$ (car la représentation transcrit R_m - cf. les propriétés p.96)

Il s'en suit :
$\forall i \in \{1 \ \ldots \ m\}$, $\forall a_i \in \mathcal{A}$, $C_i^{T_r(prop(n_0),a_i)} = C_i$
$\forall n_i \in N_{next}$, $prop(n_i) = P_{\omega_i}$ tel que $\forall i \in \{1 \ \ldots \ m\} \ \exists a_i \in \mathcal{A} \ / \ T_m(\Omega_{init}, a_i) = \omega_i$
$\forall n_i \in N_{next}$, $(n_0,n_i) \in R_{branch} \Leftrightarrow (\omega_{init}, \Omega_{prop(n_i)}) \in R_m$.
Ainsi d'après, la définition de R_a, $(n_0,n_i) \in R_{branch} \Leftrightarrow (n_0,n_i) \in R_a$.

– Récurrence :
Soit $l \in \mathbb{N}$
soit$N_l \subset N \ / \ N_l = \bigcup_{k \in \{0,\ldots,l\}} N_{branch}^k$
Supposons que $\forall (n_i,n_j) \in N_l \text{x} N_l$, $(n_i,n_j) \in R_{branch} \Leftrightarrow (n_i,n_j) \in R_a$.
Nous allons montrer que cela reste vrai pour N_{l+1}.
Par hypothèse de récurrence, $\forall n \in N_{branch}^l$, $\exists \omega \in \Omega$ *tel que* $prop(n) = P_\omega$. $\forall n_l \in N_{branch}^l$, soit C_l la classe associée à n_l. Par construction, nous avons $C_l = prop(n_l)$.

$$\exists C_{next} \subset 2^{\mathcal{C}} \text{ tel que } C_{next} = \bigcup_{a \in \mathcal{A}} \mathcal{F}_{branch}(C_l,a)$$

Soit N_{next} l'ensemble des nœuds associés: $N_{next} = \{n_1, n_2, \ \ldots \ , n_m\}$.
On a alors :

$$\forall n_i \in N_{next}, prop(n_i) = \bigcup_{P_j \in prop(n_l) \ a \in \mathcal{A} \ / \ T_r(P_j,a) \neq *} C_i^{T_r(P_j,a)}$$

$$\forall n_i \in N_{next}, (n_l,n_i) \in R_{branch}$$

Or, $\forall a \in \mathcal{A}$, $T_r(prop(n_l),a) = P_{T_m(\Omega_{prop(n_l)},a_i)} si \ T_m(\Omega_{prop(n_l)},a) \neq *$ (car la représentation transcrit R_m) - cf. les propriétés p.96)

Il s'en suit :
$\forall i \in \{1 \ \ldots \ m\}$, $\forall a_i \in \mathcal{A}$, $C_i^{T_r(prop(n_l),a_i)} = C_i$
$\forall n_i \in N_{next}$, $prop(n_i) = P_{\omega_i}$ tel que $\forall i \in \{1 \ldots m\} \ \exists a_i \in \mathcal{A} \ / \ T_m(\Omega_{prop(n_l)},a_i) = \omega_i$
$\forall n_i \in N_{next}$, $(n_l,n_i) \in R_{branch} \Leftrightarrow (\Omega_{prop(n_l)}, \Omega_{prop(n_i)}) \in R_m$.
Ainsi, d'après la définition de R_a, $(n_0,n_i) \in R_{branch} \Leftrightarrow (n_l,n_i) \in R_a$.■

Il est trivial de constater que ces deux cas particuliers sont loins d'être les deux seuls cas possibles. La figure 6.19 reprend l'exemple de *Robby* et montre plusieurs exemples de morcellements possibles. Ces possibilités ne sont pas toutes associées à une sémantique déterminée. Certaines peuvent même empêcher de trouver une solution au problème. Elles montrent la flexibilité de notre formalisation.

Il est important de noter qu'en structurant l'espace de recherche, le morcellement a aussi une influence sur son contenu. Ainsi, suivant le morcellement mis en œuvre, la relation d'atteignabilité rendra possible ou non tel ou tel enchaînement d'actions, rendra atteignable ou non tel ou tel ensemble de propriétés - que ce soit souhaitable ou non.

Propriété : Le morcellement peut empêcher des actions d'apparaître dans le graphe de planification

Preuve :

Soit $a_1 \in \mathcal{A}$ une action.
Soient $\{p_1, \ldots, p_i, p_{i+1}, \ldots, p_n\} \in 2^P$ les préconditions de a_1. Soit $\mathcal{C} = \{C_1, C_2\}$ l'ensemble de classes du morcellement considéré tel que
$\forall C_v \in \mathcal{C}, \mathcal{F}_{morc}(\{p_1, \ldots, p_i\}, C_v) = \{C_1\}$
$\forall C_w \in \mathcal{C}, \mathcal{F}_{morc}(\{p_{i+1}, \ldots, p_n\}, C_w) = \{C_2\}$
alors pour tout noeud n du graphe de planification, $T_r(prop(n), a_1) = *$
L'action a_1 n'est jamais applicable.

Même si toutes les préconditions de l'action a_1 apparaissent dans le graphe de planification, elles ne sont jamais dans une même classe. Alors que l'action a_1 peut apparaître dans un graphe de planification purement disjonctif, elle ne peut pas apparaître dans ce graphe de planification à morcellement intermédiaire.
■

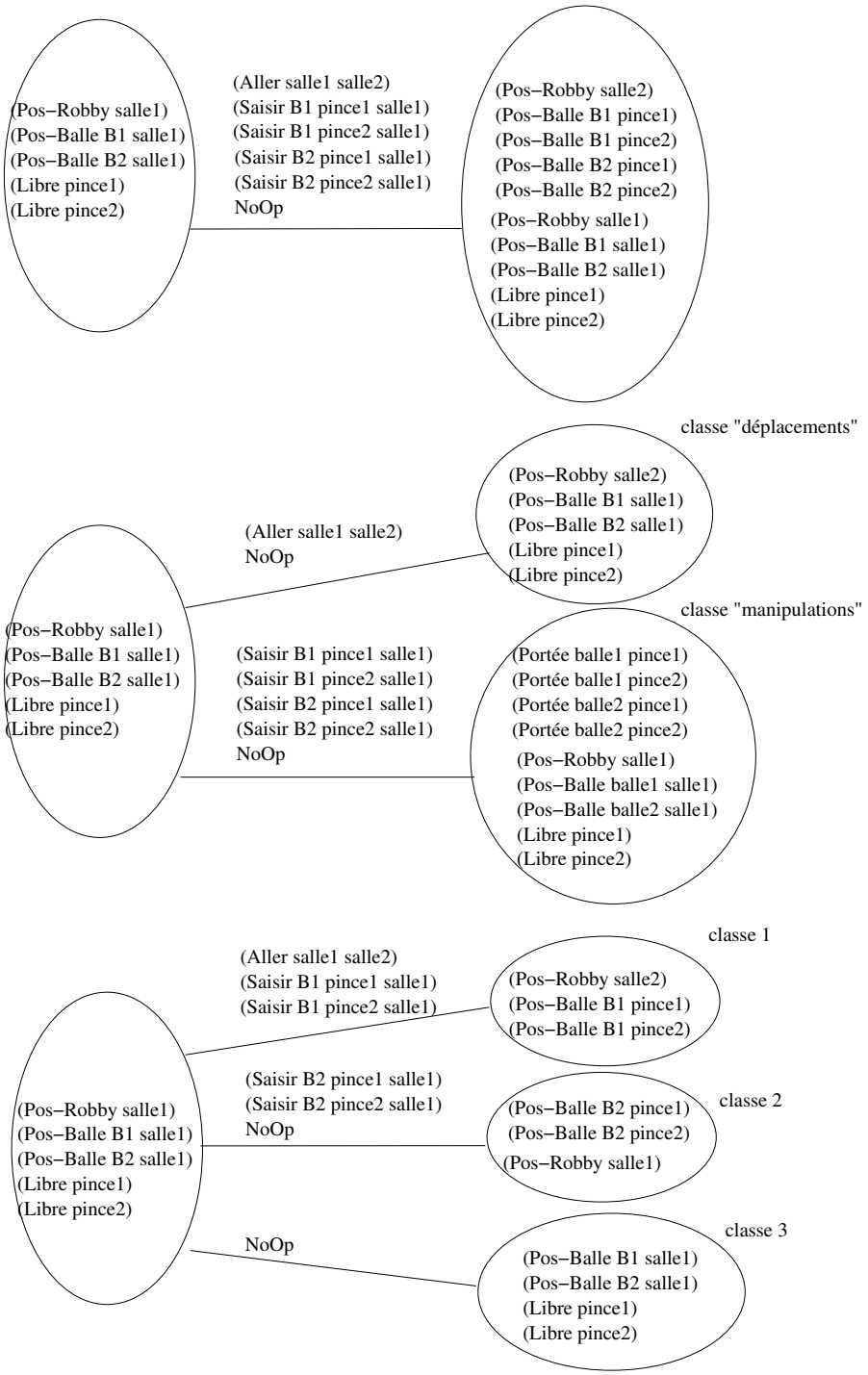


FIG. 6.19 – *Voici deux morcellements intermédiaires appliqués à un problème impliquant Robby. En haut, le morcellement nul est rappelé. Au milieu, le premier morcellement intermédiaire présenté sépare d'une part les actions de déplacement, et d'autre part les actions de manipulation des balles. En bas, le second morcellement intermédiaire présenté construit mécaniquement des classes de trois ou quatre propositions. Notons que cette stratégie de morcellement crée ici des classes stériles pour la résolution du problème. En effet, la classe* 3 *ne contient plus la position de Robby. Dans cette branche, il sera impossible de déplacer les balles vers une autre pièce.*

Propriété : Le morcellement peut empêcher des propositions d'apparaître dans le graphe de planification

Schéma de preuve :

Dans la configuration exhibée précédemment, les effets de l'action a_1 ne peuvent pas apparaître dans le graphe de planification. Soit $p \in P$ un effet de a_1 qui n'est l'effet d'aucune autre action et qui n'est pas dans l'état initial, alors p n'apparaîtra jamais dans ce graphe de planification à morcellement intermédiaire alors qu'elle serait apparue dans un graphe de planification purement disjonctif.

■

Propriété : Le morcellement peut empêcher des propositions ou des actions de ne pas être mutuellement exclusives

Schéma de preuve :

Soient $p_1 \in P$ et $p_2 \in P, p_1 \neq p_2$.

Soient trois actions a_1, a_2, et a_3 telles que le seul couple mutuellement exclusif soit (a_2, a_3) et que p_1 soit un effet de a_1 et a_2 et p_2 un effet de a_3.

Dans le graphe de planification à morcellement intermédiaire utilisé pour démontrer les deux propriétés précédentes, p_1 et p_2 sont nécessairement mutuellement exclusives car a_1 n'apparaît jamais.

Dans un graphe de planification purement disjonctif, elles pourraient ne pas être mutuellement exclusives dès l'apparition de l'action a_1.

■

Voici à présent deux exemples où le morcellement permet d'implémenter deux sémantiques précises que nous avons en tête a priori. Considérons un problème de calcul d'itinéraire. A chaque étape nous pouvons distinguer toutes les destinations atteignables en métro, celles atteignables en bus, celles atteignables en voiture par l'autoroute, celles atteignables en voiture par les routes nationales, etc. En créant une classe par mode de transport, ceux-ci sont distingués les uns des autres. Cependant dans chaque classe (chaque branche), les calculs d'itinéraires dans le réseau correspondant (lignes de métro, lignes de bus, ...) sont accomplis de manière disjonctive.

Pour prendre un exemple plus proche de la technique de planification, nous pouvons aussi penser utiliser un morcellement intermédiaire pour encoder les relations binaires de mutuelle exclusion. Deux propositions mutuellement exclusives ne devant pas être considérées simultanément dans un plan, elles seraient isolées chacune dans une branche différente (cf.figure 6.20). Nous reviendrons plus loin sur cet exemple car nous l'avons en partie intégré à notre planificateur au travers d'une implémentation spécifique.

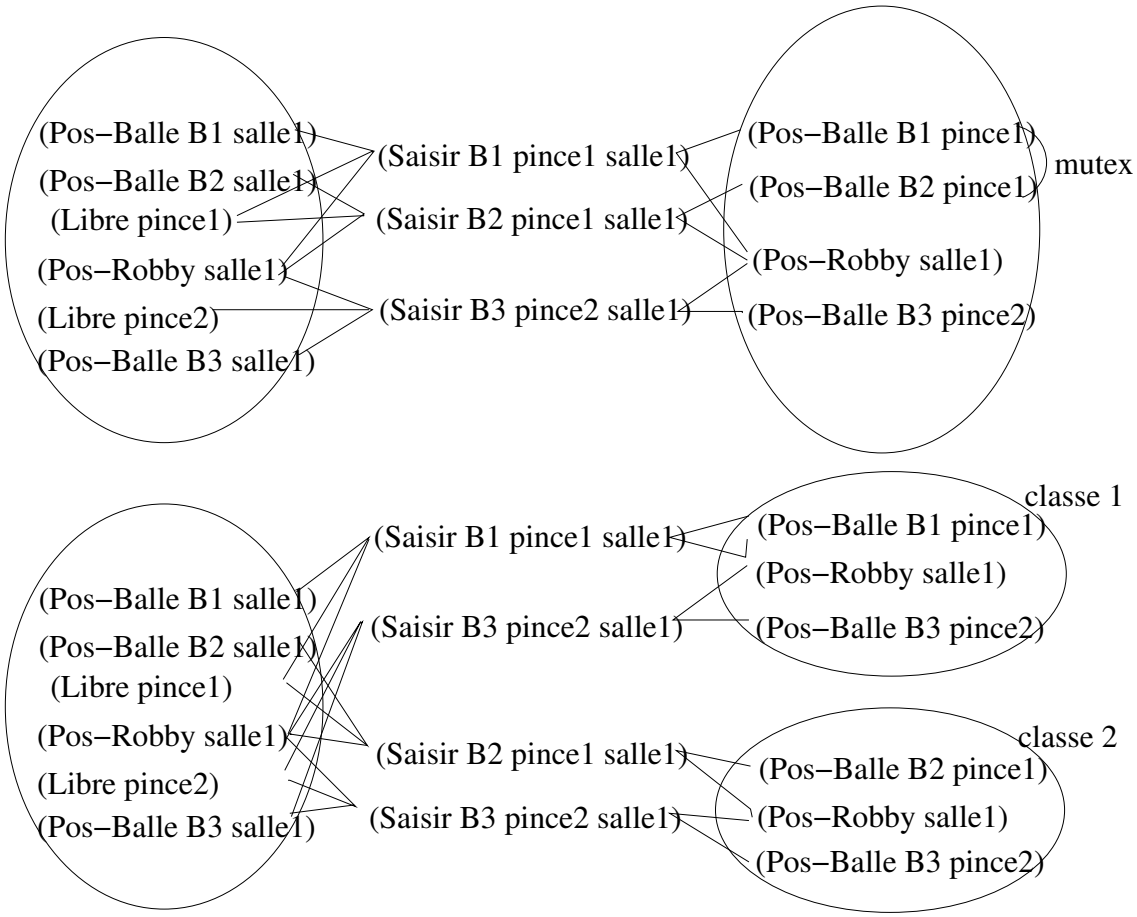


FIG. 6.20 – *Voici un morcellement intermédiaire qui encode les relations binaires de mutuelle exclusion dans la structure même de l'espace de recherche. En haut est présenté le morcellement nul (les actions NoOp sont omises). Une relation* mutex *apparaît car dans ce domaine la même pince pince*1 *ne peut pas saisir simultanment deux balles. Les deux propositions concernées ne pourront donc pas apparaître ensemble à ce niveau dans un plan solution. Le morcellement intermédiaire présenté en bas les sépare dans deux classes. C'est une autre façon d'assurer qu'elles n'apparaîtront pas ensemble à ce niveau dans un plan solution.*

6.4 Conclusion

Notre formalisation de la construction de l'espace de recherche décrit de manière cohérente les approches en apparence très différentes que sont l'approche arborescente et l'approche disjonctive à la `Graphplan`. Elle montre que ces approches ne diffèrent en fait que d'une *fonction de morcellement* qui associe à tout ensemble de propriétés un recouvrement de cet ensemble formé de *classes* de propriétés.

Il apparaît par ailleurs que de nombreuses fonctions de morcellement sont possibles. Ainsi, les deux cas des approches arborescente et disjonctive ne sont que deux cas particuliers parmi tant d'autres.

Sous jacentes à cette multitude de possibilités, plusieurs relations retiennent notre attention. D'une part, il y a la relation entre arborescence, duplication d'information dans les nœuds, et distinction de nœuds similaires mais atteints suivant des cheminements différents. D'autre part, il y a dualité entre le degré d'arborescence et la complexité des objets manipulés - et donc la taille ainsi que la forme des ensembles d'états manipulés. L'expressivité sémantique de ces relations nous laisse pressentir que certaines formulations de la fonction de morcellement peuvent tirer parti de la structure du problème à traiter. Par exemple, elles peuvent tenir compte d'une fonction d'utilité associée aux états ou aux transitions entre états, ou bien être adaptées à l'accessibilité simultanée ou alternative de certains ensembles d'états.

Chapitre 7

Le planificateur `TokenPlan`

`TokenPlan` est le planificateur que nous avons développé afin de pouvoir mettre en pratique diverses politiques de morcellement de l'espace de recherche ([54] [21] [55]). Ici, la stratégie de morcellement devient une donnée du planificateur. `TokenPlan` nous offre le contrôle nécessaire pour mettre le morcellement de l'espace de recherche au centre des choix définissant l'approche de planification à mettre en œuvre pour tel ou tel problème.

Comme `Graphplan`, `TokenPlan` construit un graphe de planification explicite qui encode l'espace de recherche et utilise deux phases alternantes de construction et de recherche. La prise en compte de la notion de morcellement de l'espace de recherche demande dans `TokenPlan` la prise en compte d'une plus grande richesse d'information. Il en découle une adaptation des mécanismes de planification en particulier au niveau de la représentation, des relations de mutuelle exclusion et des relations *no-op*.

En outre, `TokenPlan` peut être considéré comme une plateforme d'expérimentation permettant la prise en compte d'une plus grande expressivité du langage de description des domaines de planification. Il n'a malheureusement pas pu être testé sur le langage PDDL2 ([25]) qui n'était pas publié au moment de nos travaux. Cependant, nous avons amélioré l'expressivité de PDDL1 [51] en lui intégrant des fonctionnalités complémentaires suivant notre propre syntaxe.

La mise en œuvre spécifique du morcellement de l'espace de recherche est traitée dans ce chapitre à partir de la section 7.2.2. Cependant, cette mise en pratique de notre formalisation a nécessité d'adapter à un niveau plus global les représentations utilisées par le planificateur. Nous commençons par présenter les choix que nous avons fait dans ce but.

7.1 Les réseaux de Petri pour représenter les classes

Comme nous l'avons présenté dans le chapitre précédent (cf. 6.3.4), la notion de *classes* de propriétés est au centre de notre contrôle du morcellement de l'espace de recherche. Ainsi, la mise en œuvre de ce contrôle du morcellement de l'espace de recherche implique que le planificateur puisse distinguer à la fois :

1. les différentes propositions ;
2. les propositions identiques de classes différentes.

Dès lors, les propositions et les classes sont nécessaires pour étiqueter les nœuds du graphe (alors que dans le cas de la planification disjonctive, la proposition seule suffisait comme

étiquette). C'est pour cette raison que notre planificateur ne manipule pas directement les propositions, mais des entités qui portent ces deux types d'informations.

Nous avons choisi d'utiliser les réseaux de Petri, formalisme très répandu pour l'étude des systèmes à événements discrets, qui nous offrent le *jeton* comme entité pour porter les couples (proposition, classe).

Nous décrivons ci-après ce formalisme, puis la manière dont `Tokenplan` transcrit un domaine de planification en un réseau de Petri.

7.1.1 Réseaux de Petri

Un réseau de Petri se compose de *places* et de *transitions*. Une troisième entité, les *jetons*, peuvent circuler dans ce réseau.

Définition : Place et place marquée

Une place est un réceptacle à jetons - par défaut sans limitation de capacité.
Lorsqu'une place contient un jeton ou plus, elle est dite *marquée*.

Définition : Transition et transition déclenchable

Les transitions relient les places les unes aux autres. Ainsi, chaque transition a un certain nombre de places connectées en entrée et un certain nombre de places connectées en sortie.
Lorque toutes les places en entrée d'une transition sont marquées, alors la transition concernée est dite *déclenchable*.

Lorsqu'une transition déclenchable est effectivement déclenchée, chaque place en entrée perd un jeton, et chaque place en sortie en gagne un. La figure 7.1 montre cette mécanique sur un exemple.

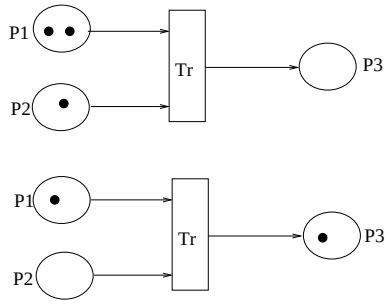


FIG. 7.1 – *Un réseau de Petri basique, composé d'une seule transition Tr et de trois places $P1$, $P2$, et $P3$. Dans la figure du haut, $P1$ et $P2$ sont marquées par des jetons (points noirs). Dans celle du bas, la transition a été déclenchée : un jeton de chaque entrée a été consommé, et la place sortie $P3$ est marquée à son tour par un jeton. Notons qu'il reste un jeton dans $P1$ mais que, $P2$ étant vide, la transition ne peut plus être déclenchée. Dans des réseaux plus complexes, les places peuvent être connectées à plusieurs transitions.*
Les places sont généralement représentées par des ellipses, et les transitions par des rectangles.

Définition : Marquage

L'ensemble des positions des jetons à un instant donné s'appelle un *marquage*.

7.1.2 Conversion d'un domaine PDDL en un réseau de Petri

Les réseaux de Petri sont conçus pour la représentation de systèmes dynamiques à événements discrets. Or, un domaine de planification contient exclusivement des contraintes de transitions discrètes. Pour cette raison, les réseaux de Petri sont adaptés à la représentation d'un domaine de planification [27], comme nous le montrons ci-après.

Considérons d'abord des opérateurs entièrement instanciés. Une place est associée à chaque proposition possible. Une transition est associée à chaque opérateur instancié. Chaque transition est connectée en entrée aux places correspondant aux préconditions, et en sortie aux places correspondant aux effets (cf. figure 7.2). Un état est représenté par le marquage de ce

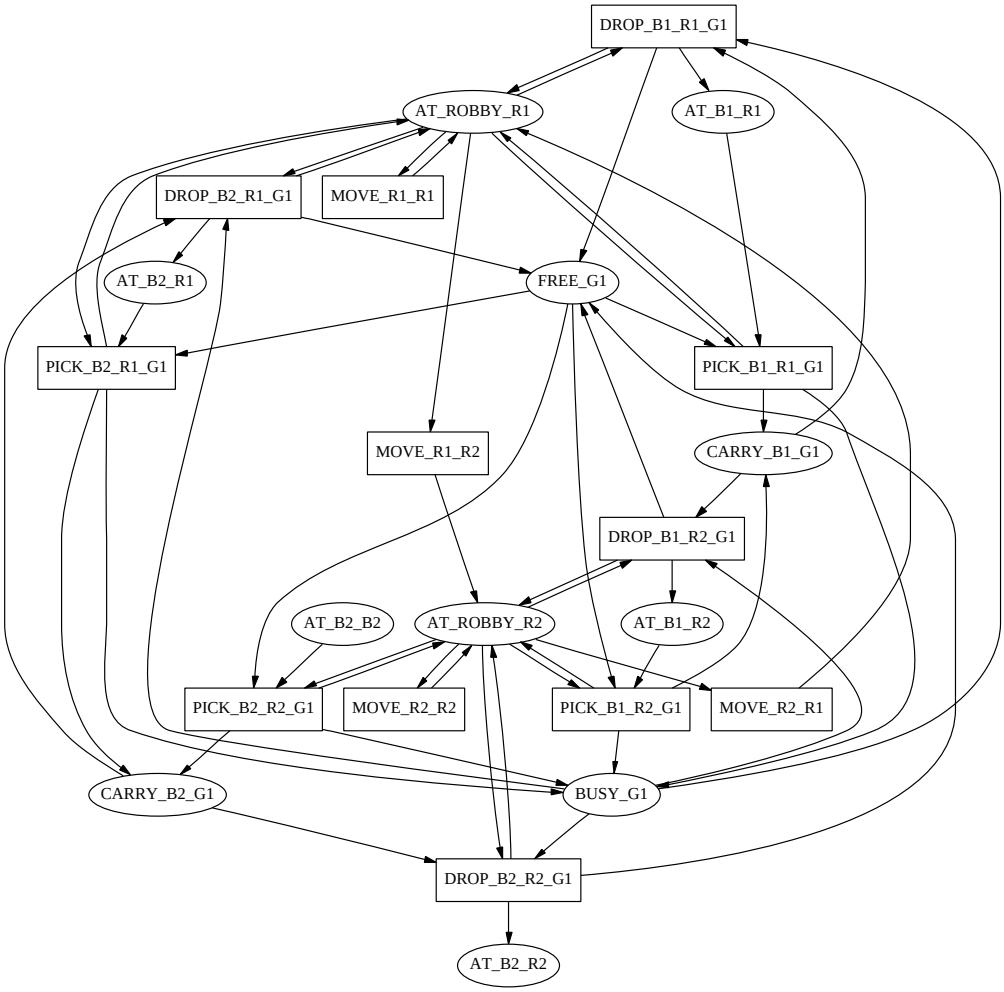


FIG. 7.2 – *Réseau de Petri correspondant au domaine du* Gripper. *Nos exemples mettant en jeu Robby en sont inspirés. Dans cet exemple, les opérateurs ont été instanciés en considérant une pince* $G1$*, deux balles* $B1$ *et* $B2$*, et deux salles* $R1$ *et* $R2$.

réseau par des jetons : si une place est occupée par un jeton, alors la proposition correspondante appartient à l'état, et réciproquement. A chaque instant, le marquage indique l'état courant et les transitions possibles à partir de cet état.

Un tel graphe peut être rendu plus compact en considérant les opérateurs non instanciés (cf. fig.7.3). Dans ce cas, lorsqu'un jeton occupe une place, il faut pouvoir distinguer quelle instanciation de la proposition associée à la place est effectivement marquée. A cette fin nous utilisons des labels : le jeton porte comme label l'assignation des variables correspondant à l'instanciation désignée. Naturellement, les transitions possibles dépendent alors aussi des labels des jetons en jeu.

Par exemple, si *Robby* se trouve dans la cuisine, la place $(AtRobby\ ?P)$ sera marquée par un jeton portant pour label $?P = cuisine$.

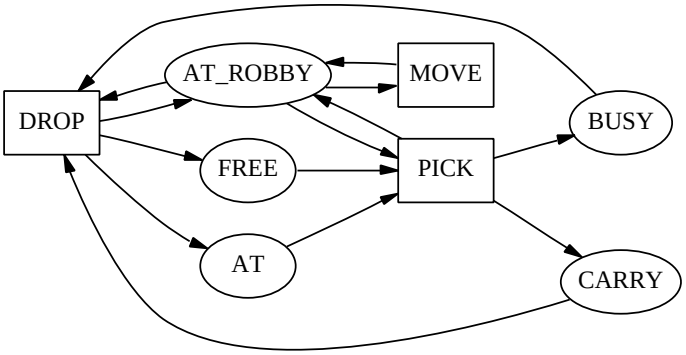


FIG. 7.3 – *Réseau de Petri à labels généré à partir du domaine du* Gripper *(le même que celui de la figure 7.2). Ce graphe, bien plus compact, représente des opérateurs non instanciés. Les instances pertinentes seront portées par les labels des jetons.*

Le déclenchement d'une transition ne résulte pas forcément en un simple mouvement de jetons, mais peut aussi causer la modification des labels portés. Ces modifications peuvent être d'ordre syntaxique, issues de raisonnements symboliques. C'est le cas pour effectuer les substitutions de variables et de valeurs qui sont propres à la planification classique (lorsqu'un jeton passe d'une place précondition à une place effet au travers d'une transition donnée). Par exemple, lorsque la transition $(move\ cuisine\ chambre)$ est déclenchée, le jeton qui marquait la place $(AtRobby\ ?P)$ avec pour label $?P = cuisine$, va retourner dans cette même place avec pour label $?P = chambre$ (voir aussi la figure 7.4).

Ces modifications peuvent aussi être numériques. Elles permettent alors une plus grande expressivité pour décrire le domaine de planification. La figure 7.4 présente un exemple.

En pratique, la transcription d'un domaine décrit suivant la syntaxe PDDL en un réseau de Petri s'effectue comme suit. A chaque *prédicat* correspond une place dans le réseau de Petri. A chaque *opérateur* (appelé `action` dans la syntaxe PDDL) correspond une transition.

Relier les préconditions aux transitions est trivial. En ce qui concerne la connexion des effets, trois catégories doivent être considérées :

1. effets positifs
2. effets négatifs
3. effets implicites

Les premiers sont eux aussi triviaux à connecter.

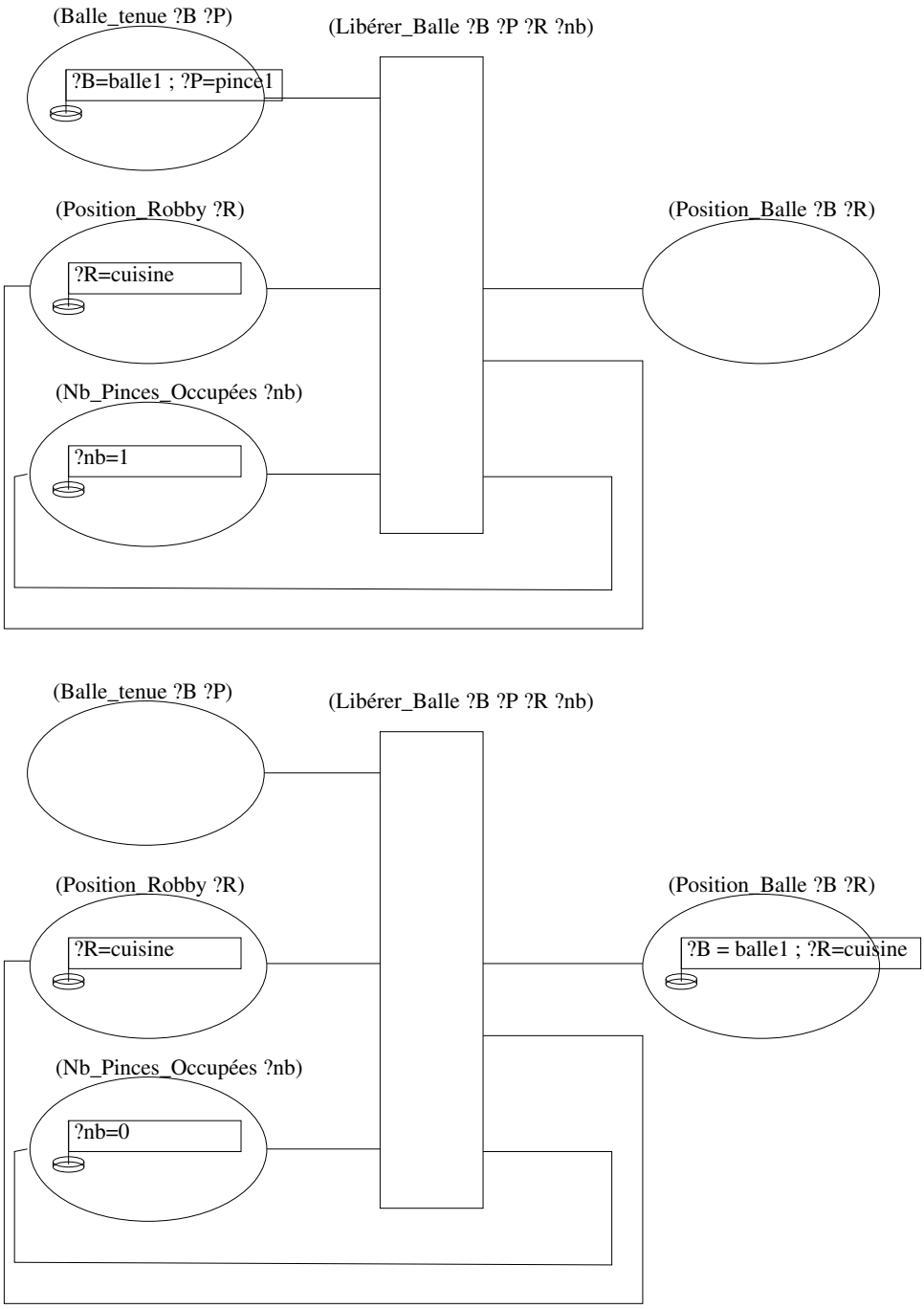


FIG. 7.4 – *Le schéma du haut présente une transition et les places qui lui sont connectées, avant son déclenchement. Le schéma du bas présente le résultat du déclenchement. Chaque drapeau correspond au label porté par le jeton correspondant. Certains de ces labels ont été modifiés lors du déclenchement de la transition.*

Les deuxièmes sont juste la manifestation du déplacement effectif de certains jetons. Un tel effet précise simplement qu'une précondition n'est plus valide après exécution de l'action. Dans le cadre du réseau de Petri cela signifie que le jeton qui marquait la place précondition a été déplacé vers une autre place (un des effets positifs). En conséquence, les effets négatifs ne modifient pas la structure existante du réseau.

En revanche, les effets implicites nécessitent de nouveaux liens. Ils relient la transition aux préconditions qui n'ont pas été détruites, c'est-à-dire aux propositions qui apparaissent dans les préconditions de l'opérateur mais qui n'apparaissent pas dans ses effets négatifs (d'où l'appellation d'effets *implicites*) (cf. figure 7.5).

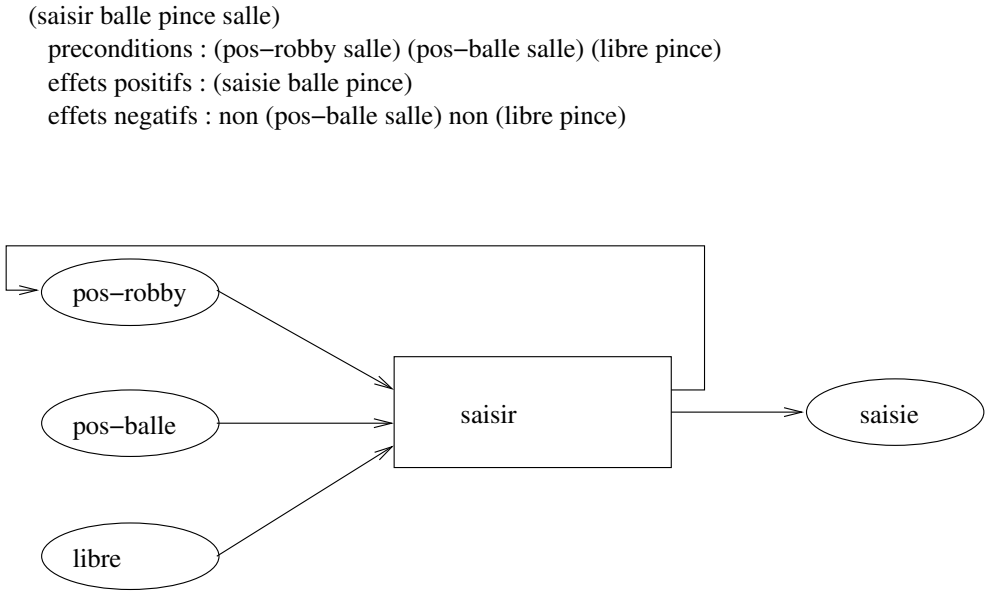


FIG. 7.5 – *Considérons en détail l'opérateur Saisir. Une description en intension est présentée. Les trois types d'effets sont représentés : deux effets négatifs, un effet positif, et un effet implicite. Celui-ci correspond à la position de Robby. Il est nécessaire que le robot et la balle soient dans la même pièce - d'où la présence de* $(pos - robby\ salle)$ *dans les préconditions. Cependant, saisir la balle ne modifie pas la position de Robby. Dans le réseau de Petri, cela se traduit par un lien retour vers la place correspondante.*

Pour chaque lien du réseau, les unifications entre les variables de la transition d'une part, et les variables de la place d'autre part, sont calculées. Elles seront nécessaire pour la modification adéquate des labels portés par les jetons lors de la planification.

Notons que le résultat obtenu est indépendant du problème : c'est bien le domaine lui-même qui a été converti. Ainsi, tout pré-traitement du réseau de Petri permettant d'améliorer par la suite les performances de planification présente un intérêt (puisqu'il pourra n'être effectué qu'une seule fois par domaine - pour tous les problèmes à traiter).

7.2 Construction du graphe de planification avec la propagation des jetons

Le réseau de Petri représentant le domaine considéré est utilisé par `TokenPlan` pour construire l'espace de recherche. Cet espace de recherche est représenté par un graphe de planification explicite.

Dans un premier temps, nous laissons de côté les contraintes liées au traitement des classes de jetons. Cela revient à traiter le cas du morcellement nul, puisqu'il correspond à l'unicité de la classe des jetons. Ainsi, le lecteur pourra s'appuyer sur ce qu'il connaît de la planification classique et plus particulièrement de `Graphplan` pour comprendre comment `TokenPlan` construit son graphe de planification à partir du réseau de Petri qui représente le domaine de planification.

Dans un second temps, la possible multiplicité des classes est introduite. Cette présentation progressive de l'implémentation souligne d'une part le respect de la cohérence de la modélisation des domaines, et d'autre part la flexibilité du morcellement de l'espace de recherche.

7.2.1 Le graphe de planification mémorise les transitions dans le réseau de Petri

Un problème classique de planification se compose d'un domaine, d'un état initial et d'un ensemble d'objectifs. `TokenPlan` construit l'espace de recherche en avant, à partir de l'état initial.

Première étape : marquage de l'état initial

`TokenPlan` utilise le marquage du réseau de Petri pour décrire ce qui est atteignable. La première étape est donc de recréer le marquage de l'état initial à l'aide de jetons et du réseau de Petri du domaine.

Propagation des jetons fondée sur leur rémanence

Avec ce marquage, plusieurs transitions peuvent éventuellement être déclenchées. `TokenPlan` déclenche systématiquement les transitions qui peuvent l'être. Cependant, il est omis d'enlever les jetons des places préconditions après qu'ils aient été consommés.

En effet, une place marquée signifie qu'elle est atteignable. Or, ce qui a été atteint en N coups, l'est toujours au $N+1$ème coups. Cette "omission" joue un rôle semblable à celui des actions *NoOp* dans `Graphplan`.

Ainsi, toutes les transitions peuvent être déclenchées. La figure 7.6 montre un exemple avec deux transitions.

Fusion des jetons

Si la place destination d'un jeton contient déjà un jeton portant le même label, alors les deux jetons sont fusionnés.

Avec ce comportement, le marquage du réseau ne correspond plus à un état. Comme le sont les niveaux de `Graphplan`, ici chaque marquage est une approximation de l'union des états qui pourraient être marqués. Il s'agit donc de l'union des états atteignables, en première approximation.

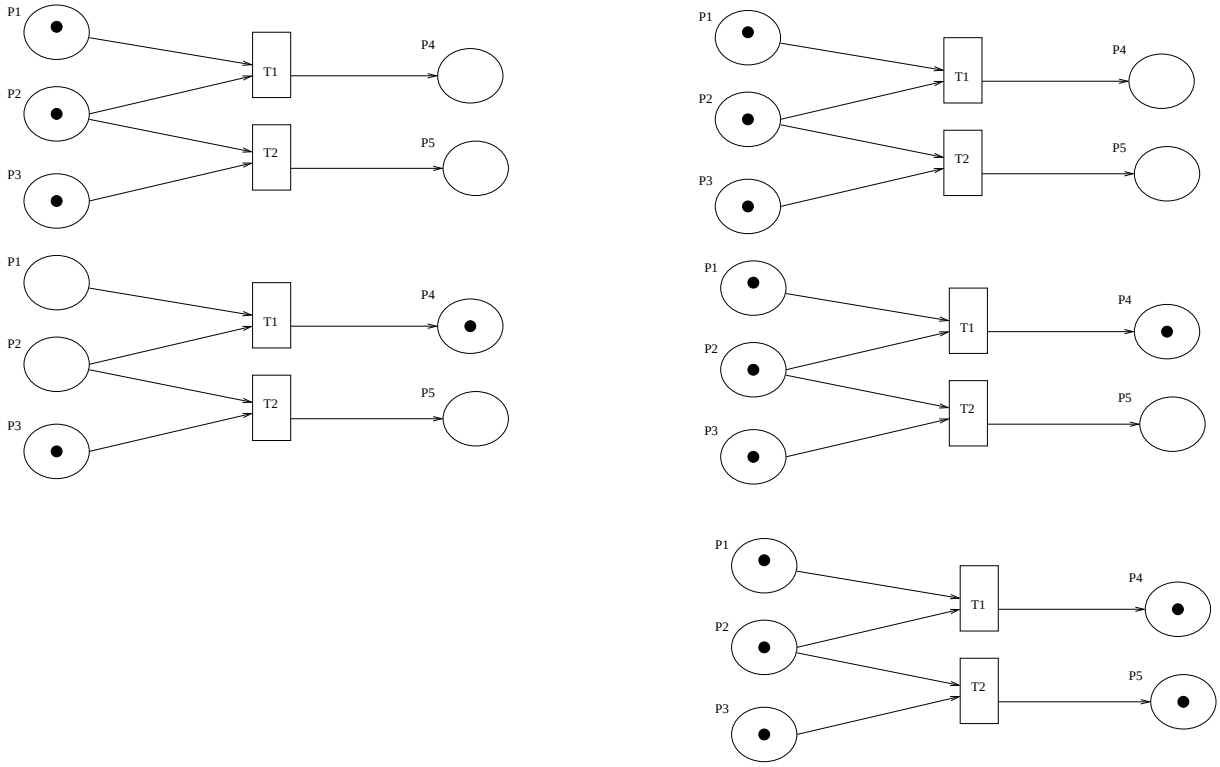


FIG. 7.6 – *A gauche est présenté le comportement classique d'un réseau de Petri. Les deux transitions $T1$ et $T2$ sont déclenchables. En déclenchant $T1$, les deux jetons marquant les préconditions sont consommés. Après ce déclenchement, $T2$ n'est plus déclenchable car il n'y a plus assez de jetons. A droite est présenté le comportement utilisé au sein de* `TokenPlan`*. Lorsque $T1$ est déclenchée, un nouveau jeton est introduit dans la place de son effet positif, mais les jetons de ses préconditions ne sont pas détruits. En conséquence, la transition $T2$ peut à son tour être déclenchée.*

Mémorisation de l'évolution du marquage

Ce que nous venons de présenter n'est pas suffisant pour construire l'espace de recherche. En effet, pour pouvoir chercher un plan, il faut aussi mémoriser l'évolution elle-même du marquage. Pour ce faire, deux types d'informations sont mémorisées : d'une part certains marquages, et d'autre part les transitions en jeu pour passer des uns aux autres.

Le marquage initial est mémorisé. Toutes les transitions qui y sont déclenchables sont déclenchées. Le nouveau marquage est mémorisé. A nouveau, toutes les transitions déclenchables le sont, et le marquage suivant est mémorisé. Ainsi, le premier marquage est obtenu sans transition (c'est l'état initial). Le deuxième nécessite éventuellement plusieurs transitions en parallèle - mais aucune séquence. Le troisième est atteint par une séquence de deux groupes de transitions (au sein d'un groupe toutes les transitions sont en parallèle), etc...

Structures des jetons

Pour mémoriser plus aisément les transitions utilisées, des *jetons de transitions* sont créés. Ils correspondent aux éléments des niveaux "Actions" du graphe de `Graphplan`.

Définition : *Jetons de transition*

Le mouvement d'un jeton au travers d'une transition est décomposé en deux. D'une part le mouvement d'une place précondition vers la transition. D'autre part, de la transition vers une place effet. Cette décomposition sert à manipuler plus aisément les modifications induites par le passage d'une transition. Dans ce cadre-là, lorsque nous parlerons de l'étape au sein d'une transition nous parlerons *d'un jeton de transition.*

Le label d'un jeton de transition porte les assignations des variables de l'opérateur. Par exemple, lorsque l'action ($Aller\, cuisine\, chambre$) est utilisée, la transition associée à l'opérateur ($Aller\, ?salle1\, ?salle2$) reçoit un jeton ayant pour label $<?salle1 = cuisine\ ?salle2 = chambre>$.

Deux raisons nous ont menés à distinguer les jetons de transitions des autres jetons :

1. Les réseaux de Petri, que nous utilisons comme base de notre représentation, n'ont pas de marquage sur les transitions. Il est donc difficile de pister les jetons à ce moment-là sans introduire les jetons de transition.
2. Dans le cadre des mécanismes de `TokenPlan`, les labels des jetons de transitions porteront des informations concernant les classes sensiblement différentes de celles portées par les autres jetons (nous y revenons dans la section suivante).

Par ailleurs, chaque jeton porte deux listes de jetons qui correspondent à ses provenances et à ses destinations. Ainsi, le jeton marquant la précondition ($posRobby\, cuisine$) - J_1 - a pour destination le jeton marquant l'action ($Aller\, cuisine\, chambre$) - J_2. J_2 a dans sa liste de provenances le jeton J_1. Pour destination il a un jeton J_3 qui marque la proposition ($posRobby\, chambre$). Bien sûr, J_1 peut avoir d'autres destinations, suivant les pièces que peut atteindre *Robby* à partir de la cuisine. De la même manière, il existe peut-être plusieurs façons d'atteindre la chambre. J_3 aura une liste de jetons de provenance d'autant plus longue. Les jetons qui sont à une place identique d'un marquage à l'autre sont reliés par des actions de copie - que nous appellerons *NoOp* puisqu'elles ont le même rôle que dans `Graphplan`.

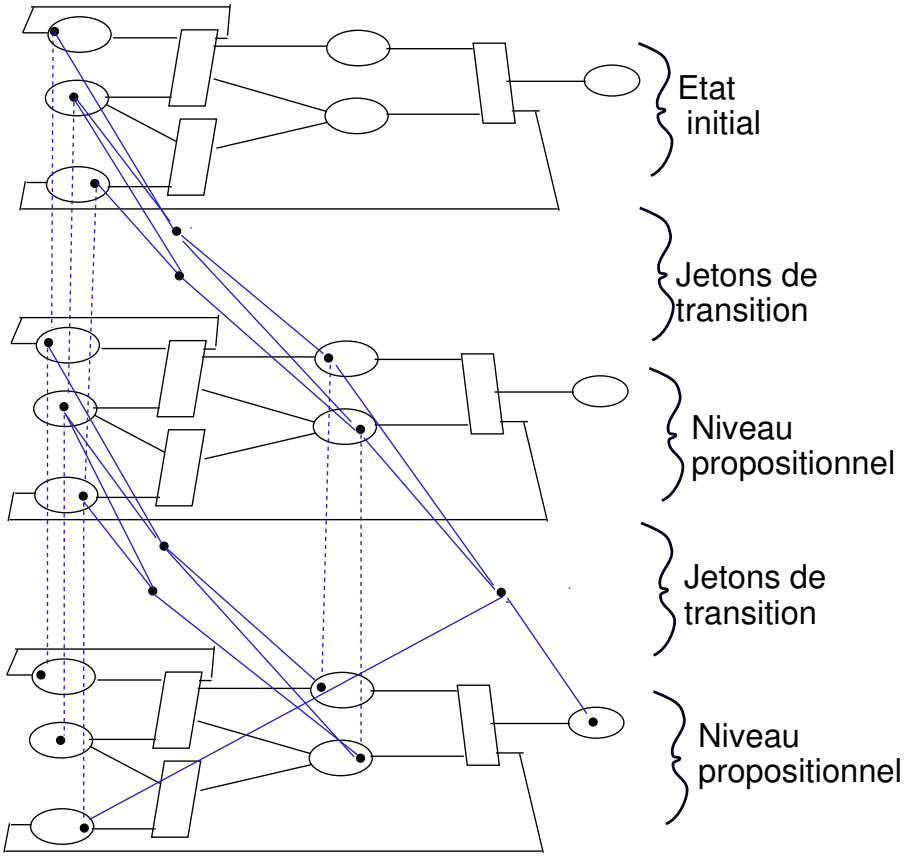


FIG. 7.7 – *Le marquage initial est mémorisé (réseau de Petri du haut). Deux transitions sont déclenchées. Le marquage résultant est mémorisé à son tour (réseau de Petri en dessous). Les mouvements des jetons sont mémorisés eux aussi. Les liens correspondants sont verticaux car ils relient des mémorisations successives du marquage du réseau de Petri. Entre deux réseaux de Petri, nous pouvons noter les jetons de transition. En pointillés figurent les actions* NoOp. *La propagation des jetons se poursuit de la même manière " empilant " les réseaux de Petri les uns sous les autres. Si l'on ne considère que les liens verticaux et les jetons, nous obtenons un graphe organisé en niveaux dans lequel alternent les niveaux de jetons associés à des propositions et les niveaux de jetons de transitions.*

Récapitulons. Chaque jeton porte les informations suivantes :

- Label : assignations des variables (et informations sur ses classes d'appartenance)
- liste de provenances : liste des jetons de provenances
- liste de destinations : liste des jetons de destinations

Le label d'un jeton est associé au réseau de Petri, c'est-à-dire à la place ou à la transition qu'il marque. Les deux listes de provenances et de destinations le relient aux autres jetons, et de manière duale à son passé et à son avenir. Lorsque le jeton considéré marque une place, ces listes contiennent des jetons de transitions. Dans ce cas la liste de *provenances* correspond au *support* de la proposition associée. Lorsqu'il s'agit d'un jeton de transition, elles contiennent des jetons marquant des places.

La propagation des jetons résulte en un graphe organisé en niveaux

Le processus de propagation des jetons s'apparente au mécanisme de "dépliage" des réseaux de Petri et se fait au travers de plusieurs copies du réseau de Petri. Chaque copie est comme une photo du marquage après une étape, deux étapes, trois étapes, etc. De l'une à l'autre, des liens relient les jetons de sorte que l'évolution elle même est mémorisée. La figure 7.7 représente ce processus par une sorte d'empilement de réseaux de Petri. Le graphe de planification qui représente l'espace de recherche est constitué des liens entre les jetons. Il s'agit d'un sous-graphe d'une sorte de "dépliage" des réseaux de Petri (cf. figure 7.8). Il est organisé en niveaux et il est semblable à celui de `Graphplan` lorsqu'aucune relation de mutuelle exclusion n'est calculée. Ce résultat est cohérent puisque nous avons présenté le cas du morcellement nul.

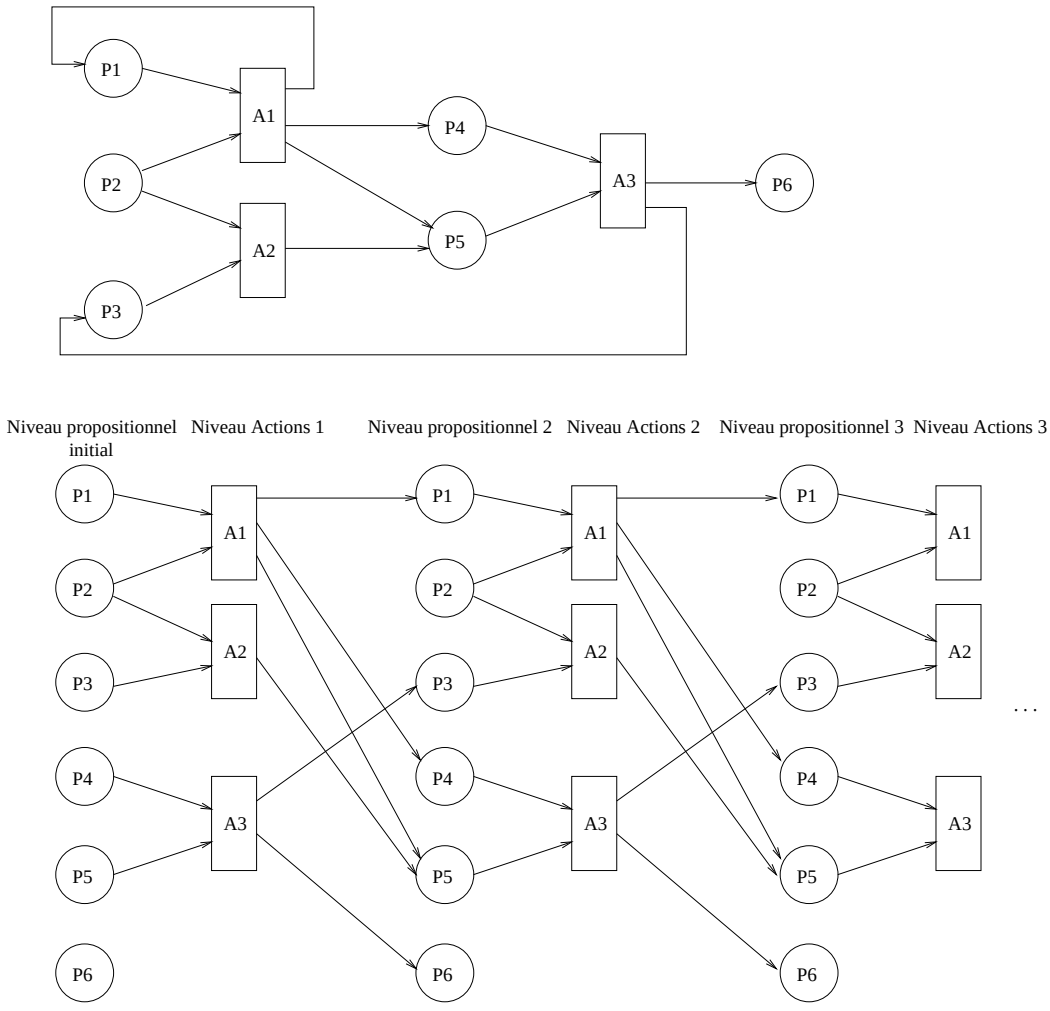


FIG. 7.8 – *Le schéma du bas représente un "dépliage" sur trois niveaux du réseau de Petri du schéma du haut. Le graphe de mémorisation de la propagation des jetons s'inscrit dans ce "dépliage". Nous obtenons un graphe constitué de niveaux propositionnels alternant avec des niveaux d'actions, comme celui construit par* ***Graphplan****.*

7.2.2 Filtrage des transitions déclenchées en fonction des classes des jetons

Notre modélisation du morcellement de l'espace de recherche est fondée sur le concept de *classes* (cf. section 6.3.4). Nous avons vu que les jetons portent tous des informations concernant ces mêmes classes. Nous décrivons ici comment ces informations sont exploitées pour transcrire le morcellement de l'espace de recherche dans le graphe de planification.

La clef de voûte de cette transcription est une contrainte de déclenchement des transitions liée aux classes des jetons qui marquent les places préconditions des transitions.

Contrainte de déclenchement liée aux classes :
Pour être déclenchable, une transition doit avoir toutes ses places-préconditions marquées par des jetons non mutuellement exclusifs **appartenant tous à la même classe**.

Cette contrainte traduit simplement le fait que chaque branche - donc chaque classe de jetons - est traitée indépendamment dans les calculs d'atteignabilité.

Par ailleurs, les classes sont des signes distinctifs des jetons. Ainsi, deux jetons marquant une même place, portant le même label, mais de classes différentes, seront considérés comme différents et ne seront pas fusionnés. C'est comme s'ils étaient apparus dans deux branches distinctes d'un arbre de recherche et que l'on ne souhaite pas confondre ces deux manières d'y arriver. C'est ce qui permet de différencier deux instances d'une même proposition obtenues par deux enchaînements d'actions différents, par exemple avec des coûts ou des utilités différents.

La classe des effets dépendra de cette classe commune aux préconditions et des directives de morcellement associées à la transition considérée. A la place d'une classe, le jeton de transition correspondant porte l'ensemble des transitions de classes. Il s'agit de la liste des classes atteignables avec cette transition, et pour chacune d'entre elles les classes dans lesquelles il faut être au départ. La figure 7.9 montre un exemple de transition, et les classes associées à tous les jetons.

Techniquement, les informations portées par les jetons peuvent être factorisées à divers degrés et de plusieurs manières. Ainsi, trois implémentations sont possibles :

1. Tous les jetons différents sont considérés et manipulés individuellement ;
2. Les jetons ne différant que par les assignations de variables que portent leurs labels et marquant une même place sont fusionnés en un seul. Son label porte l'ensemble des assignations pertinentes. Il s'agit d'un rassemblement par classe.
3. Les jetons ne différant que par leurs classes d'appartenance et marquant une même place sont fusionnés en un seul. Son label porte l'ensemble des classes d'appartenances pertinentes. Il s'agit d'un rassemblement par proposition.

C'est cette troisième option qui a été retenue au sein de `TokenPlan`. Les classes étant représentées par des entiers, les jetons portent de simples ensembles d'entiers. Il s'agit d'une facilité d'écriture qui permet de manipuler en une seule fois les jetons de toutes les classes concernées.

En particulier, pour qu'une transition soit déclenchable il suffit que l'intersection des ensembles de classes des jetons qui marquent ses préconditions ne soit pas vide. La transition est alors déclenchable dans les classes de cette intersection. Les mécanismes d'unification entre les variables des préconditions et celles de l'action sont accomplis une seule fois pour toutes

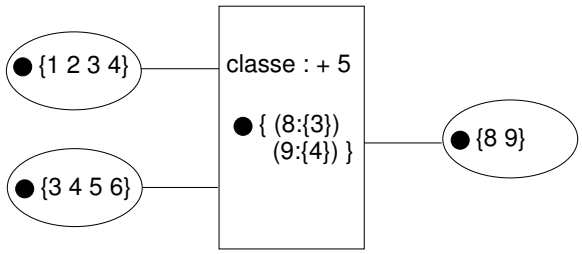


FIG. 7.9 – *Voici un exemple de transition, avec deux places préconditions et une place effet. Cette transition a une directive de changement de classe : si les préconditions sont dans une classe C, les effets sont dans la classe $C + 5$. Les deux places préconditions sont marquées chacune par un jeton. Les classes de chaque jeton sont indiquées à sa droite. La transition peut être déclenchée dans les classes 3 et 4 car ce sont celles que les jetons ont en commun. Il s'ensuit que l'effet est présent dans les classes 8 (3+5) et 9 (4+5). L'information concernant les classes portée par le jeton de transition indique que la classe 8 peut être atteinte par cette transition à partir de la classe 3, et la classe 9 à partir de la 4. La formulation dans ce sens peut surprendre. En fait elle est adaptée à la recherche arrière - qui est l'utilisatrice de ces informations.*

les classes en présence (c'est-à-dire pour toutes les branches de l'arbre de planification). Les classes des effets de cette transition dépendent de cette intersection (cf. fig. 7.9).

7.3 Relations de mutuelle exclusion

L'approximation de l'étude d'atteignabilité peut ici aussi être améliorée par la prise en compte de relations de mutuelle exclusion. `TokenPlan` considère seulement les relations binaires de mutuelle exclusion. Après avoir reformulé les règles de mutuelle exclusion en termes de jetons, nous discutons des difficultés introduites par le morcellement intermédiaire.

7.3.1 Relations de mutuelle exclusion entre jetons de même classe

Les règles de calcul des relations de mutuelle exclusion présentées page 61 peuvent se reformuler en termes de jetons comme suit.

Règle d'interférence

Si une transition "consomme" un jeton (c'est-à-dire qu'elle ne le remet pas dans sa place d'origine) alors elle est mutuellement exclusive avec toutes les transitions dont l'une des places précondition était marquée par ce jeton ou qui amènent le même jeton (même label) dans la même place (effet positif).

Règle de compétition de ressources

Si un jeton J_1 marque une place précondition d'une transition T_1, et un jeton J_2 marque une place précondition d'une transition T_2, et si J_1 et J_2 sont mutuellement exclusifs, alors T_1 et T_2 sont elles aussi mutuellement exclusives.

Cas des jetons marquant des places

Deux jetons marquant une place sont mutuellement exclusifs si tous les jetons de transition de leurs supports respectifs sont mutuellement exclusifs deux à deux.

7.3.2 Morcellement et relations de mutuelle exclusion

Les relations permanentes de mutuelle exclusion entre jetons de transition ne dépendent pas des classes car elles sont intrinsèques aux actions. Toutes les autres dépendent de la classe considérée. Deux jetons peuvent être mutuellement exclusifs dans une classe et ne pas l'être dans une autre.

Ces différences sont liées aux différences entre les supports d'un même jeton d'une classe à l'autre. Si les supports sont identiques, les relations de mutuelle exclusion sont elles-aussi identiques.

Pour simplifier le traitement, `TokenPlan` ne tient pas compte des classes pour le calcul et la mémorisation des relations de mutuelle exclusion. Il considère des *relations de mutuelle exclusion transclasses*. Ce choix réduit le nombre de cas à traiter lorsqu'est considéré un jeton présent dans plusieurs classes. Cela réduit aussi l'espace mémoire nécessaire au stockage des relations *mutex*.

Relations de mutuelle exclusion transclasses

Lorsque les relations de mutuelle exclusion entre jetons ne portent que sur les propositions ou actions correspondantes et ne tiennent pas compte de leurs classes, nous parlerons de *Relations de mutuelle exclusion transclasses.*

Calculer des relations transclasses revient à mélanger tous les supports (donc à mélanger les branches). Ainsi, deux jetons liés par une relation de mutuelle exclusion transclasses sont mutuellement exclusifs dans toutes leurs classes communes. En revanche, deux jetons qui sont mutuellement exclusifs dans certaines classes peuvent ne pas être repérés par les relations transclasses.

Il est important de noter que calculer les relations transclasses seulement ne fait rien perdre à la complétude du planificateur. En effet, d'une part aucun couple de jetons n'est considéré mutuellement exclusif à tort, et d'autre part les relations de mutuelle exclusion non détectées se retrouvent lors de l'extraction du plan.

Enfin, le nombre de ces relations de mutuelle exclusion non détectées par les relations transclasses dépend de l'ampleur des différences entre les ensembles de transitions déclenchables d'une classe à l'autre pour des propositions communes. Considérons l'exemple extrême où deux classes contiennent les mêmes propositions mais aucune action en commun. Il est alors probable qu'il y ait des relations d'exclusion mutuelle locales aux classes. Elles ne seront pas détectées par la version actuelle de `TokenPlan` qui se contente des relations transclasses. Ainsi, la proportion de ces relations de mutuelle exclusion locales aux classes dépend de la structuration du problème et de la stratégie de morcellement mise en œuvre.

7.4 Actions *NoOp* et morcellement

En ce qui concerne les propositions portées par les jetons, les actions *NoOp* restent de simples opérateurs de copie. En revanche, en ce qui concerne les classes portées par les jetons, la simple copie dans le niveau suivant de ce que contient le niveau courant n'est plus suffisante.

Comportement des actions *NoOp* en cas de morcellement :

En cas de morcellement, les actions *NoOp* doivent :

- copier l'existant (l'introduction de nouvelles classes n'implique pas l'abandon des classes courantes) ;
- copier dans les nouvelles classes les jetons existants **sauf** ceux dont les propositions associées sont détruites par toutes les actions menant à cette classe ;
- indiquer à quelles transitions de classes elles correspondent (pour la recherche arrière).

Considérons un exemple. Supposons que le niveau courant décrive un état (comme en morcellement total). Par exemple, *Robby* est dans la cuisine et porte deux balles. Cet état est décrit par des jetons de classe 0. Le robot peut aller soit dans le salon, soit dans le jardin.

Tous les jetons de la classe 0 doivent bien-sûr être copiés au prochain niveau. C'est pour "copier l'existant" qui perdure.

Les deux actions sont distinguées par le morcellement, qui crée une nouvelle branche pour chacune. L'effet $(Pos - Robby\, salon)$ sera porté par un jeton de classe 1. L'effet $(Pos - Robby\, jardin)$ sera porté par un jeton de classe 2. Cependant, dans un cas comme dans l'autre *Robby* porte toujours deux balles. Les jetons correspondants doivent donc être présents dans les deux nouvelles classes. C'est "copier dans les nouvelles classes les jetons existants".

Toutefois, le jeton marquant $(Pos - Robby\, cuisine)$ ne doit pas apparaître dans les classes 1 et 2. Dans le cas contraire, les jetons de la classe 1 ne marqueraient plus un état mais un ensemble d'états (puisque deux positions seraient possibles pour le robot). Le morcellement total ne serait donc pas réalisé. Voilà pour ce qui concerne "sauf ceux (...) détruits par toutes les actions menant à cette classe".

Dans `TokenPlan` ceci est accompli selon un processus systématique, comme suit. Lors du déclenchement de toutes les transitions à un niveau donné, chaque changement de classe est mémorisé dans une table. Cette table indique pour chaque classe existante C (les classes de départ) les classes atteintes par les changements de classes du niveau considéré (en partant de C). Pour chacune de ces classes destinations, une liste de "jetons interdits" est mémorisée aussi (cf. figure 7.10).

Définition *Jetons interdits*

Un *jeton interdit* J est associé à une classe C lorsque J ne doit pas être copié dans la classe C par une action *NoOp*.

Lors du déclenchement d'une transition T faisant passer de la classe C_1 à la classe C_2, la liste des jetons interdits associés à ce changement de classe est mise à jour. Si ce changement de C_1 vers C_2 n'avait pas été référencé avant, alors tous les effets négatifs de T sont mémorisés comme jetons interdits. Si ce changement était déjà référencé, alors la liste des jetons interdits devient l'intersection entre cette liste et les effets négatifs de T.

Chaque niveau conserve une telle table des transitions de classes.

Origine	Dest.	Jetons interdits
1	2 : 3 :	J1 J2 J2 J3 J4
2	1 : 6 :	J7 J2 J5
3	∅	

FIG. 7.10 – *Voici un exemple de table de transitions de classes associée à un niveau donné. A partir de la classe* 1*, il est possible d'atteindre les classes* 2 *et* 3*. Dans le premier cas, les jetons* $J1$ *et* $J2$ *ne peuvent pas suivre la transition. Dans le second cas il s'agit des jetons* 2*,* 3*, et* 4*. Notons que cette table indique qu'aucune transition n'est possible à partir de la classe* 3*.*

7.5 Recherche d'un plan solution

Dans le cas d'un morcellement non nul, les objectifs peuvent apparaître dans plusieurs classes. Dans un parallèle avec les approches arborescentes (*FSS, A**, ...), cela correspond au fait que les objectifs sont atteints par plusieurs branches que l'on souhaite distinguer.

Condition de déclenchement de la recherche arrière

Lorsque tous les objectifs sont marqués par des jetons de même classe, non mutuellement exclusifs, alors la construction du graphe de planification est interrompue et une recherche arrière de plan est lancée.

La classe commune des jetons correspond à la branche dans laquelle les objectifs semblent atteignables. C'est cette branche qui est suivie jusqu'à la racine de l'arbre pour trouver un plan. Ainsi, `TokenPlan` impose que tous les jetons sélectionnés pour un niveau propositionnel soient dans une classe commune.

Soit C_{obj} l'intersection des ensembles de classes portés par les jetons. Si C_{obj} est vide alors la recherche s'arrête immédiatement.

Dans le cas contraire, la recherche peut se poursuivre. En choisissant les actions de soutien, deux contraintes supplémentaires doivent être prises en compte par rapport à celles de la recherche arrière standard :

1. il faut que les objectifs soient tous soutenus dans une classe commune, au sein de C_{obj} ;
2. il faut que les préconditions de toutes ces actions aient elles-aussi une classe en commun. Soit C_{prec} l'intersection - non vide - des ensembles de classes portés par les jetons de préconditions.

La satisfaction de ces contraintes est évaluée grâce aux transitions de classes portées par les jetons de transition.

Si ces deux contraintes sont satisfaites, la recherche reprend à partir des préconditions. C_{prec} est la nouvelle contrainte de classes à respecter. Bien sûr, comme dans la recherche arrière standard, il est vérifié que les actions ne sont pas mutuellement exclusives. Les échecs sont aussi mémorisés.

L'algorithme 2 décrit le choix des actions de support au sein d'un même niveau. Un autre appel récursif permet de remonter de niveau en niveau avec les valeurs de *nextButs* et de *nextClasses* retournées par cet algorithme. Elles correspondent respectivement aux propositions à soutenir dans le niveau qui va être traité, et aux classes dans au moins l'une desquelles elles doivent être soutenues. La figure 7.11 détaille un exemple de régression dans le cadre de cette recherche.

Algorithme 2: Pendant la recherche arrière il faut tenir compte des classes pour en tirer parti. Cet algorithme prend en charge l'assignation d'actions à une série de buts au sein d'un même niveau. L'appel initial se fait avec la liste des buts, des classes qui leurs sont communes, et trois ensembles vides pour les actions déjà sélectionnées, les prochains buts et les prochaines classes (pour le niveau traité ensuite).

```
AttribuerSupport(Buts, ClassesCommunes, ActionsChoisies, nextButs, nextClasses)
si Buts = ∅ alors
    retourner ActionsChoisies, nextButs, NextClasses
sinon
    but ← depiler(Buts)
    pour chaque action dans support(but) faire
        classes-prec-admissibles← {c ∈ classes(prec(action))/classes(effets(actiondansc))∩
        ClassesCommunes ≠ ∅}
        si classes-prec-admissibles ≠ ∅ et ¬ mutex(action, ActionsChoisies) alors
            AttribuerSupport(Buts, ClassesCommunes, ActionsChoisies ∪ {action},
            nextButs ∪ prec(action), nextClasses ∩ classes-prec-admissibles)
```

Le fait de considérer toujours l'ensemble des classes communes entre les jetons alors qu'une seule est nécessaire permet de progresser plus profondément dans les niveaux sans devoir rebrousser chemin (*backtrack*). Prenons pour exemple le cas de deux objectifs qui apparaissent dans deux classes C_1 et C_2. Supposons qu'en fait ils ne pourront être soutenus qu'en C_2, mais que ce n'est qu'au bout de 5 niveaux qu'il est possible de s'en apercevoir. Dans notre cas, la recherche arrière continue jusqu'au premier niveau du graphe. Si au contraire seule la classe $C1$ avait été considérée - pour commencer, alors au bout de 5 niveaux il faudrait tout reprendre à partir des objectifs, en considérant cette fois la classe $C2$.

En revanche, considérer les intersections oblige à procéder à une validation supplémentaire. Une fois que l'état initial est atteint, le plan doit être parcouru en avant pour calculer précisément les classes effectivement traversées. Pendant la recherche arrière, les ensembles de classes correspondent à des évaluations des possibilités. Le parcours supplémentaire en avant permet de préciser lesquelles sont effectives. Il est direct et sans branchement.

Par exemple, supposons que le but se compose de deux propositions appartenant toutes deux aux classes 5 et 6. Supposons ensuite que la classe 5 puisse être atteinte depuis la classe 3 et la classe 6 depuis la 4. Il est alors noté que l'objectif peut être atteint dans les classes 5 et 6, et la recherche arrière est relancée à partir du niveau précédent, en quête de préconditions appropriées dans la classe 3 ou bien la 4. Finalement, un ensemble de propositions compatibles est trouvé en classe 3. Cela remplit l'objectif courant de la recherche, qui reprend au niveau précédent, et ainsi de suite. Cependant, avec ce plan, le but n'est atteint qu'en classe 5 seulement. Le rôle de la vérification avant est d'établir avec précision les transitions de classes,

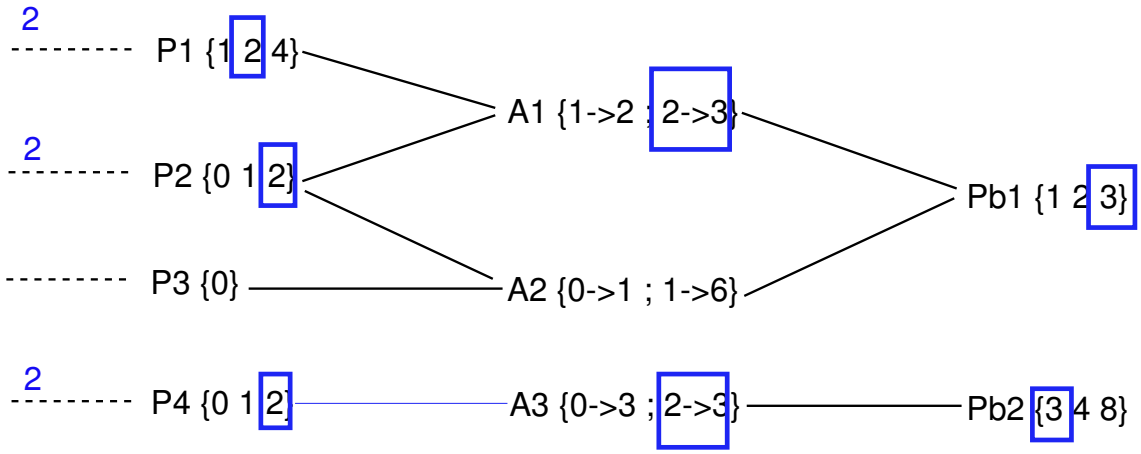


FIG. 7.11 – *Voici à titre d'exemple le début d'une recherche arrière. Deux buts doivent être soutenus : Pb*1 *et Pb*2*. Ils ont une seule classe en commun :* 3*. Ils doivent donc être soutenus dans celle-ci. Considérons le cas de Pb*1*. Deux actions la supportent. A*2 *ne peut être retenue car elle ne peut soutenir Pb*1 *que dans les classes* 1 *et* 6*. A*1*, en revanche peut soutenir Pb*1 *dans la classe* 3*. Cependant il faut pour cela qu'elle soit déclenchée dans la classe* 2*. Pb*2 *de son côté peut être soutenue dans la classe* 3 *par A*3*. A*3 *n'est pas mutuellement exclusive avec A*1*. Leurs préconditions respectives peuvent toutes êtres soutenues dans une classe commune - la classe* 2*. Enfin, elles ne sont pas mutuellement exclusives. La recherche peut donc se poursuivre à un niveau inférieur. Il faut désormais soutenir P*1*, P*2*, et P*4 *dans la classe* 2*.*

montrant dans ce cas que le but n'est pas atteint dans les deux classes 5 et 6, mais seulement dans la classe 5.

7.6 Saturation du graphe de planification et condition d'arrêt de la planification

`Tokenplan` détecte lorsqu'un problème n'a pas de plan solution, sous certaines conditions. Le graphe de planification arrive alors à "saturation". C'est-à-dire qu'il n'y a plus de changement d'un niveau à l'autre : pas de nouvelle action, pas de nouvelle proposition, les mêmes relations de mutuelle exclusion et les mêmes ensembles échec.

Comme il est montré dans [6], cette propriété repose sur le fait que les descriptions en intension génèrent des langages finis. Ceci reste vrai pour `Tokenplan`. Cependant, dans le cas d'un morcellement non nul, il faut aussi prendre en compte l'évolution des classes. Ainsi, si le morcellement génère un nombre fini de classes, le graphe de planification "saturera" aussi au niveau des classes.

Nous verrons dans le prochain chapitre, que dans le cadre de l'optimisation, nous pouvons aussi nous accomoder d'un ensemble de classes infini.

7.7 Spécification du morcellement dans le domaine de planification

`TokenPlan` accepte des domaines décrits dans le langage *PDDL*. Il supporte quelques extensions comme les domaines typés, les tests entre les paramètres des actions, ou encore le calcul d'expressions mathématiques (soit dans les tests, soit pour calculer les valeurs des effets numériques). Pour ce dernier aspect, la syntaxe *PDDL* n'a pas été respectée cependant car elle était quelque peu lourde. D'ailleurs, elle ne faisait pas le consensus. Depuis, elle a été retravaillée pour donner lieu au langage *PDDL2* ([25]) mais ceci n'a pas pu être pris en compte en raison de l'interruption de nos travaux en septembre 2001.

Avant toute chose, la spécificité de `TokenPlan` est de pouvoir appliquer à la construction de l'espace de recherche une grande variété de stratégies de morcellements. Ces morcellements sont décrits par le concepteur en même temps que le domaine de planification.

Pour spécifier le morcellement, il suffit de définir la manière de créer les classes et de passer des unes aux autres. `TokenPlan` représente les classes par des entiers et les manipule essentiellement par opérations arithmétiques. Ces opérations ont lieu dans les transitions.

Ceci permet de fonder la gestion des classes sur les trajectoires en termes d'enchainements d'actions. C'est par exemple le cas quand chaque transition doit ajouter ou soustraire au numéro de la classe des préconditions un entier plus ou moins grand. Suivant l'enchaînement d'actions pour arriver à une proposition particulière, la classe du jeton correspondant sera plus ou moins élevée.

Cependant, il est aussi possible de fonder la gestion des classes sur des ensembles d'états définis a priori. Il faut pour cela conditionner le choix de la classe aux paramètres de l'action. Par exemple, si tel effet est présent alors le numéro de la classe est tel nombre.

Ces directives sont écrites dans la description du domaine, par le concepteur. A cette fin, le langage *PDDL* a été augmenté de nouvelles instructions. Par ailleurs la syntaxe concernant les évaluations d'expressions mathématiques a été modifiée pour plus de flexibilité.

Normalement, chaque opérateur est décrit par trois champs : `:parameters` , `:precondition`, et `:effect`. Nous avons rajouté le champs facultatif `:class`. Il contient l'instruction de morcellement. Sept directives différentes sont actuellement supportées :

1. `nil` signifie qu'aucun morcellement n'est réalisé. C'est équivalent à omettre le champs `:class` ;
2. `new` signifie qu'une nouvelle classe doit être créée. Ceci est accompli en tenant à jour une liste des classes utilisées ;
3. `rand` indique de choisir aléatoirement une classe parmis celles utilisées ;
4. `precomputed` suivi d'un nombre, indique directement le numéro de la classe ;
5. `cond` permet de choisir la classe en fonction des valeurs de paramètres sélectionnés de l'action ;
6. `:split` est suivi d'une fonction f ($+$, $-$. $*$, $/$, ...) et d'un nombre n. La classe est $f(c,n)$, où c est la classe des préconditions ;
7. `:split-fct` est suivi d'une expression. La classe est le résultat de l'évaluation de cette expression. Cette expression se compose du nom d'une fonction et de la valeur de tous ses paramètres sauf du premier (qui est par convention la classe courante). Les noms de

paramètres de l'action peuvent être utilisés comme paramètres de la fonction. Ainsi, la fonction peut faire des opérations complexe sur des valeurs contextuelles, éventuellement numériques. L'implémentation Lisp rend à cet égard les choses flexibles. Le concepteur du domaine peut invoquer n'importe quelle fonction. Il lui faudra simplement la charger dans l'interpréteur Lisp avant de lancer la planification.

Au lieu d'être intégrées au domaine de planification, ces directives de morcellement pourraient être rassemblées dans un fichier séparé. `TokenPlan` prendrait alors en paramètres deux fichiers :

1. le domaine de planification ;
2. la description d'une stratégie de morcellement adaptée à ce domaine.

7.8 Conclusion

Fondé sur une approche disjonctive, le planificateur `TokenPlan` que nous avons développé permet de mettre en pratique une grande variété de stratégies de morcellement de l'espace de recherche.

Ainsi, ce planificateur n'est plus seulement paramétré par un domaine de planification, mais aussi par une stratégie de morcellement. Dans des versions ultérieures, il serait bon d'expliciter cette distinction entre ces deux paramètres, car une stratégie de morcellement peut être

- absolument générique : elle est applicable à tous les domaines de planification (ce serait par exemple le cas d'une stratégie de morcellement basée sur les relations de mutuelle exclusion) ;
- générique pour un ensemble de domaines : elle tire parti d'une spécificité propre à une catégorie de domaines (elle est par exemple liée à une notion de "navigation" qui définirait une sorte de type de domaines) ;
- propre à un domaine, mais adaptée à tous les problèmes de celui-ci ;
- adaptée à un problème particulier dans un domaine donné.

Par ailleurs, un développement possible de ces travaux est de permettre la génération automatique de stratégies de morcellement. Dans ce cas, le concepteur du domaine de planification n'est clairement pas le concepteur de la stratégie de morcellement adaptée.

Chapitre 8

Expressivité du morcellement : degré d'arborescence et plus encore

Nous avons introduit la notion de morcellement de l'espace de recherche dans le but de pouvoir contrôler le degré d'arborescence des processus de construction du graphe de planification. Cependant, il s'avère que le morcellement permet aussi des structurations de l'espace de recherche qui ne sont pas arborescentes.

Ce chapitre présente différentes structurations - les unes arborescentes, les autres non - qui correspondent aux besoins de différents types de problèmes de planification. En s'attachant à partir du problème à traiter, nous nous plaçons dans la situation de celui qui écrit un domaine de planification et qui doit décrire au planificateur le morcellement à utiliser.

Dans un premier temps nous abordons des structurations arborescentes. Ensuite, nous présentons des structurations non arborescentes. Pour ces dernières, il est nécessaire d'adapter la recherche arrière.

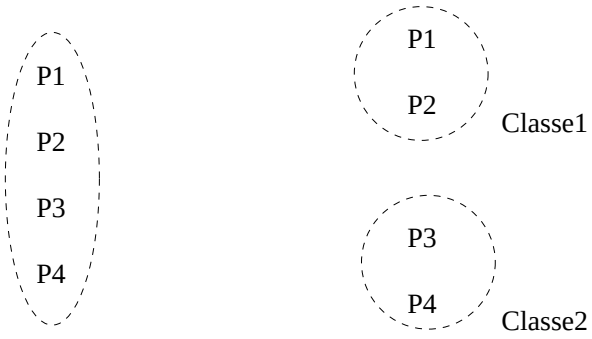
8.1 Morcellement et degré d'arborescence

Dans la lignée de ce que nous avons présenté, voici deux types de problèmes de planification qui peuvent gagner à l'introduction d'une certaine arborescence dans la planification disjonctive. Nous considérons d'abord le problème classique de planification, puis une extension de celui-ci vers le traitement des incertitudes.

8.1.1 Problème classique et relations de mutuelle exclusion

L'erreur de l'étude d'atteignabilité sous-jacente à la construction d'un espace de recherche avec un morcellement nul provient du fait que les ensembles d'états manipulés sont plus gros que les ensembles d'états effectivement atteignables. Les relations de mutuelle exclusion réduisent cette erreur en retirant de l'espace des zones invalides.

Cependant, pour retrouver un résultat exact, il faudrait calculer **toutes** les relations de mutuelle exclusion possibles : binaires, ternaires, etc. . Faire l'impasse sur certaines d'entre-elles revient à accepter un niveau d'erreur plus grand. C'est un moyen de régler le compromis entre rapidité de construction de l'espace de recherche et précision de l'étude d'atteignabilité qu'il encode.



{P1} {P2} {P3} {P4}
{P1 P2} {P1 P3} {P1 P4}
{P2 P3} {P2 P4} {P3 P4}
[P1 P2 P3} {P1 P2 P4} {P1 P3 P4}
{P2 P3 P4} [P1 P2 P3 P4}

classe1 : {P1} {P2} {P1 P2}
classe2 : {P3} {P4} {P3 P4}

FIG. 8.1 – *A gauche, un ensemble de propositions en morcellement nul. En dessous, les sous-ensembles de ces propositions qui peuvent être considérés pour l'introduction d'actions. A droite, cet ensemble a été morcelé en deux classes (deux branches). En dessous, le nombre de cas possibles pouvant être considérés pour l'introduction d'actions est fortement réduit. Toutes les propositions sont conservées par le morcellement. En revanche, l'ensemble des "états potentiels" qu'elles peuvent représenter est sensiblement réduit.*

Il en découle un autre compromis entre facilité de construction de l'espace de recherche et facilité de la recherche et de l'extraction éventuelle d'un plan solution. En ne tenant compte que des relations binaires de mutuelle exclusion, `Graphplan` atteint globalement une efficacité bien meilleure que les approches à morcellement total.

Le niveau d'exactitude de l'espace de recherche peut aussi se régler en modifiant son morcellement - cette fois à niveau fixe de mutuelle exclusion pris en compte. En effet, les relations de mutuelle exclusion peuvent être encodées par la structure de l'espace de recherche elle-même. Le morcellement total, par exemple, encode toutes les relations de mutuelle exclusion possibles.

Apparaît alors un nouveau compromis. A niveau d'erreur fixé, l'économie du calcul de relations de mutuelle exclusion recquiert l'introduction de plus de morcellement (adapté!), et vice versa. Il faut alors mettre en relation l'éventuel surcoût d'un morcellement plus fin par rapport au coût du calcul des relations de mutuelle exclusion, avec la modification du coût de la recherche de plan que cela entraîne.

Morceler à un niveau intermédiaire revient à diviser la branche courante (unique en planification disjonctive) en plusieurs branches - introduisant une certaine arborescence. En conséquence, l'ensemble d'états qui peut être défini à l'aide de toute les propositions de la branche initiale est morcelé en sous-ensembles définis par les propositions des nouvelles branches.

Alors que le morcellement de l'ensemble initial de propositions est un recouvrement, le morcellement induit de l'ensemble d'états ne l'est pas. Ainsi, certains "états potentiels" disparaissent de l'ensemble considéré, car les propositions qui les composent sont réparties entre plusieurs classes : elles ne peuvent pas être considérées ensemble (cf. figure 8.1).

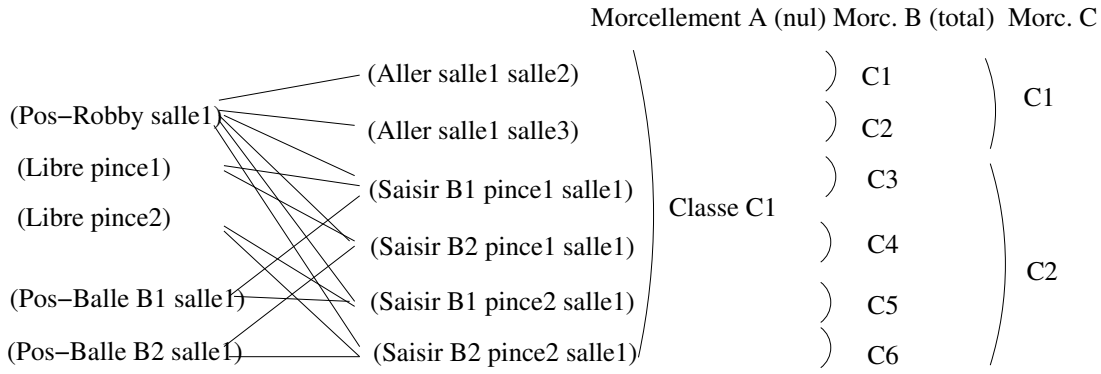


FIG. 8.2 – *Voici un premier niveau d'actions à partir d'un état initial donné. Chacune des actions de déplacement est mutuellement exclusive avec toutes les autres. En effet Robby ne peut pas être en plusieurs endroits à la fois! Certaines des actions de saisie sont aussi mutuellement exclusives entre elles (une même pince ne peut pas saisir deux balles à la fois et une même balle ne peut pas être saisie par deux pinces en même temps).*

L'idée ici est de profiter de ce redécoupage pour reprendre dans chacun des nouveaux ensembles le moins possible d'états non valides. La figure 8.2 présente trois morcellement possibles sur un même exemple. Dans le cadre du morcellement nul, 13 couples mutuellement exclusifs sont recensés. Avec un morcellement total, 6 classes sont introduites - une pour chaque action. De fait, il n'y a plus de couple mutuellement exclusif (puisqu'il n'y a plus de couple au sein d'une classe). Le morcellement C est intermédiaire. Il introduit deux classes qui séparent les actions de déplacement des actions de saisie. 8 relations de mutuelle exclusion sont encodées au travers de ce morcellement : tous les "états potentiels", en fait invalides, dans lesquels au moins une balle était saisie et *Robby* était parti dans une autre salle ne sont plus considérés. Il reste cependent un couple mutuellement exclusif dans la classe $C1$ et 4 dans la classe $C2$:

1. $(Saisir\ B1\ P1)$ et $(Saisir\ B2\ P1)$;
2. $(Saisir\ B1\ P2)$ et $(Saisir\ B2\ P2)$;
3. $(Saisir\ B1\ P1)$ et $(Saisir\ B1\ P2)$;
4. $(Saisir\ B2\ P1)$ et $(Saisir\ B2\ P2)$.

Autre exemple : si tous les "états potentiels" laissés de côté correspondent à ceux contenant des triplets de propositions mutuellement exclusives, alors il n'est plus nécessaire de calculer les relations ternaires de mutuelle exclusion. Celles-ci sont encodées dans la structure de l'espace de recherche.

Dans le cadre des approches de planification heuristique, la quête d'heuristiques de plus en plus précises est de première importance. Nous avons vu comment des variantes de `Graphplan` sont utilisées à cette fin. Dans ce contexte, trouver des critères de morcellement faciles à mettre en œuvre pour encoder à moindre coût tout ou partie des relations de mutuelle exclusion dans la structure de l'arbre peut être une piste intéressante à explorer. [77] décrit des travaux dans cette voie.

Bien-sûr, l'aspect indépendant de chaque branche est ici important. Si les branches se retrouvaient mélangées lors de la recherche d'un plan, nos efforts pour écarter certains états invalides seraient perdus.

8.1.2 Extension du problème classique à la planification dans l'incertain

Considérons un problème similaire à celui traité par `Sensory Graphplan` dans [74]. Toutes les actions sont déterministes, mais l'état initial est incertain. Toutefois, les différents états initiaux possibles sont connus. Dans le problème spécifique de [74], deux variables booléennes ont une valeur incertaine, mais seuls deux des quatre états éventuels sont considérés comme possibles.

En développant deux graphes indépendants pour chaque état initial éventuel, `Sensory Graphplan` (cf. section 4.4.3 de la partie 1), met en pratique un morcellement intermédiaire. Deux branches indépendantes sont créées. Cependant, à l'intérieur de chacune d'entre elles, c'est un morcellement nul qui est utilisé. Ce faisant, les deux autres cas possibles par combinatoire - mais impossibles d'après les connaissances associées au domaine - ne sont pas envisagées.

Dans le cadre de notre présentation du morcellement de l'espace de recherche, pour résoudre le problème considéré, il serait question de créer deux classes - une par état initial éventuel (cf. figure 8.3). Toutes les propositions de l'état initial appartiendraient aux deux classes à la fois, sauf celles correspondant aux deux variables incertaines. Pour chacune de ces variables, chaque instanciation possible ne serait présente que dans une région seulement.

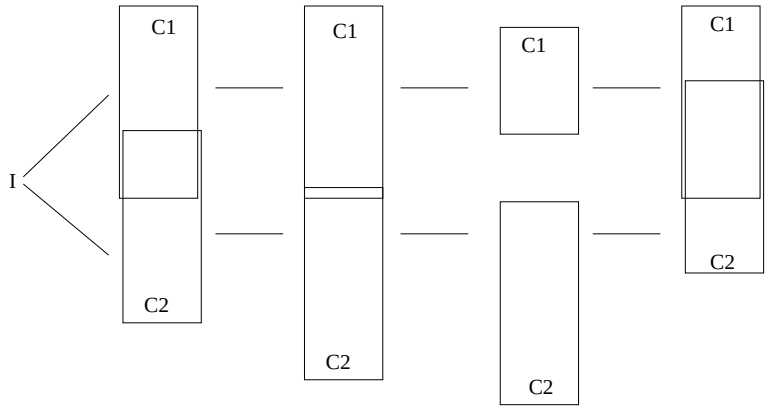


FIG. 8.3 – *Le niveau initial est réparti en deux classes : une pour chaque état initial possible. Ces classes se recoupent plus ou moins. S'il est fait abstraction des actions NoOp, cette zone de recouvrement est plus ou moins présente d'un niveau à l'autre. Elle peut être totalement absente de certains niveaux. Dans ce cas, il n'existe aucune action qui soit exécutable dans les deux situations.*

Il faut aussi s'assurer qu'aucune action du domaine ne permette de changer de classe. Ainsi, une séquence d'actions débutant dans la classe de l'un des états initiaux possibles, s'exécute entièrement et se termine dans cette même classe.

Plans conformants

Un plan conformant correspond à une suite d'états présente dans la zone de recouvrement de toutes les classes (ici seulement deux sont considérées). Il est exécutable indifféremment quelle que soit la réalité effective.

Plans contingents

Malheureusement, toutes les classes ne se recoupent pas forcément de manière appropriée. Dans ce cas les frontières des zones de recouvrement utilisées par un plan renseignent quant aux observations que doit contenir ce plan. D'une part, elles indiquent en partie l'information qu'il faut obtenir (ce qu'il faut pouvoir distinguer). D'autre part, elles indiquent l'étape du plan avant laquelle il est impératif d'avoir obtenu ladite information. Si des actions d'observation satisfaisantes existent et peuvent être inclues dans le plan aux endroits appropriés, alors ce plan est un plan contingent. Ici, les différentes classes sont liées à la notion de *contexte* présente dans `C-Buridan` par exemple. Les zones de recouvrement correspondent aux parties compatibles des contextes.

Actions non-déterministes

Si les actions elles-mêmes ne sont pas déterministes, alors chaque exécution peut mener à un état incertain dont on connait plus ou moins les diverses alternatives a priori. Le morcellement utilisé précédemment peut être généralisé à ce cas : chaque effet possible d'une action correspond alors à une nouvelle branche.

Bien-sûr, le morcellement peut devenir très grand. L'espace de recherche devient alors du même coup très lourd à gérer. Ce sont les mêmes critiques que celles avancées face à une généralisation possible de l'approche de [74] dans les mêmes circonstances. Toutefois, elles peuvent être amoindries s'il est possible de tirer parti des zones de recouvrement entre les branches. `Sensory-Graphplan`, tout comme `Conformant Graphplan`, développe des graphes disjoints pour chaque possibilité. Il s'ensuit que le travail concernant les parties communes des graphes est réitéré autant de fois qu'il y a de possibilités pour sa partie variable. Pour éviter cela, il faut ne pas dupliquer les parties communes, de sorte que le graphe de planification reflète le fait que le morcellement définit des zones qui se recouvrent. Chaque jeton manipulé par `TokenPlan` porte l'ensemble des classes auxquelles il appartient. Ce choix algorithmique va dans le sens d'une mise en évidence des recouvrements du morcellement.

8.2 Autres critères de morcellement

Dans ce qui suit, nous sortons de la problématique d'introduire ou non de l'arborescence dans les approches disjonctives. Nous ne conservons que la notion brute de morcellement de l'espace de recherche. Il s'agit de définir des zones de l'espace de recherche qui forment un recouvrement de cet espace.

Dans un premier exemple, cette structure transcrit des connaissances de haut niveau que le concepteur possède à propos de la tâche de planification à traiter. Le second type de problèmes abordé concerne l'extension du problème classique de planification vers l'optimisation.

Dans un cas comme dans l'autre, il n'y a plus de notion de branche de recherche qu'il suffit de remonter des objectifs vers l'état initial pour expliciter un plan solution. Il est donc impératif de modifier la recherche arrière.

8.2.1 Filtrage dans le cadre de problèmes riches

Certains domaines de planification peuvent avoir de nombreuses actions différentes pour lesquelles le déclenchement est peu contraint. La construction de l'espace de recherche peut

alors devenir problématique compte tenu de la charge de travail. Dans les faits, cependant, ces actions ne sont peut-être pas utiles partout où elles peuvent être exécutées.

Cette information peut être connue du concepteur. Il peut la transmettre au planificateur en imposant un certain morcellement et en n'autorisant certaines actions que dans certaines régions de l'espace de recherche. Le morcellement devient alors un moyen de représentation de métastructures du domaine.

Morcellement de filtrage

Il est question d'une stratégie de *morcellement de filtrage* quand les opérateurs du domaine de planification contiennent parmi leurs préconditions des contraintes de classes. Ces contraintes forcent le planificateur à ne pas introduire l'action dont les préconditions ne sont pas réunies dans l'une des classes qu'elles autorisent. Ce procédé permet de filtrer une partie des introductions possibles des actions.

Il est important de noter qu'un morcellement de filtrage ne concerne plus des préoccupations d'arborescence. En effet, l'état initial peut contenir plusieurs classes pour permettre l'entrée en jeu de plusieurs sous-ensembles d'actions. Toutes ces actions contribuent à la construction d'un plan. Elles ne sont donc pas cloisonnées dans des branches distinctes.

Un premier exemple

Considérons l'écriture d'un domaine adapté à un ensemble d'agents hétérogènes, en tirant parti du morcellement de l'espace de recherche. Les actions de tel ou tel type d'agent peuvent être distinguées par l'attribution de contraintes de classes spécifiques. Cette différenciation par les classes peut être mise à profit pour choisir une recherche de solutions en utilisant tous les types d'agents ou seulement certains, par exemple.

Un second exemple

Une stratégie de morcellement de filtrage peut aussi servir à réintroduire une sorte de structuration de coopération. Prenons l'exemple de deux agents différents dont le second a un rôle de support au premier. Il est intéressant d'éviter l'insertion systématique de tout le jeu d'actions alors que celles du second agent n'existent que pour apporter une aide au premier agent lorsque la situation le nécessite. Les actions du second agent peuvent recquérir en préconditions une classe qui n'est pas présente dans l'état initial. Au cours de la planification, le premier agent peut générer des ensembles d'états de cette classe afin de faire entrer en jeu son partenaire dans des situations particulières.

8.2.2 Extension du problème classique à l'optimisation

Dans le cadre de l'élaboration d'un plan optimal, le morcellement de l'espace de recherche doit être adapté à la fois à la "mécanique d'enchaînement des actions" et à l'évaluation des valeurs d'utilité. L'idée est de redécouper les grands ensembles d'états du morcellement nul en ensembles plus petits et homogènes du point de vue de l'utilité.

Cela ré-introduit une certaine redondance dans le cadre de la planification disjonctive, mais, idéalement, pas plus qu'il n'est nécessaire pour permettre d'optimiser un critère. La figure 8.4 permet de mieux comprendre la démarche. D'une part, il y a les ensembles d'états

définis par le morcellement nul de la planification. D'autre part, il y a les ensembles d'états définis par un morcellement fondé sur le regroupement des état de même utilité.

Puisque nous sommes intéressés ici par l'optimisation en planification, le morcellement adéquat va correspondre au produit cartésien des deux. Compte tenu des outils à notre disposition, nous choisirons la représentation de l'espace de recherche de la planification classique et nous y introduirons le morcellement basé sur les valeurs d'utilité. Cependant, notons qu'il serait possible de se placer dans l'espace défini précédemment pour l'optimisation et d'y inclure le morcellement lié aux structures de chaînage des actions (sous réserve qu'il soit possible de remonter à cette structure à partir des seules valeurs d'utilité).

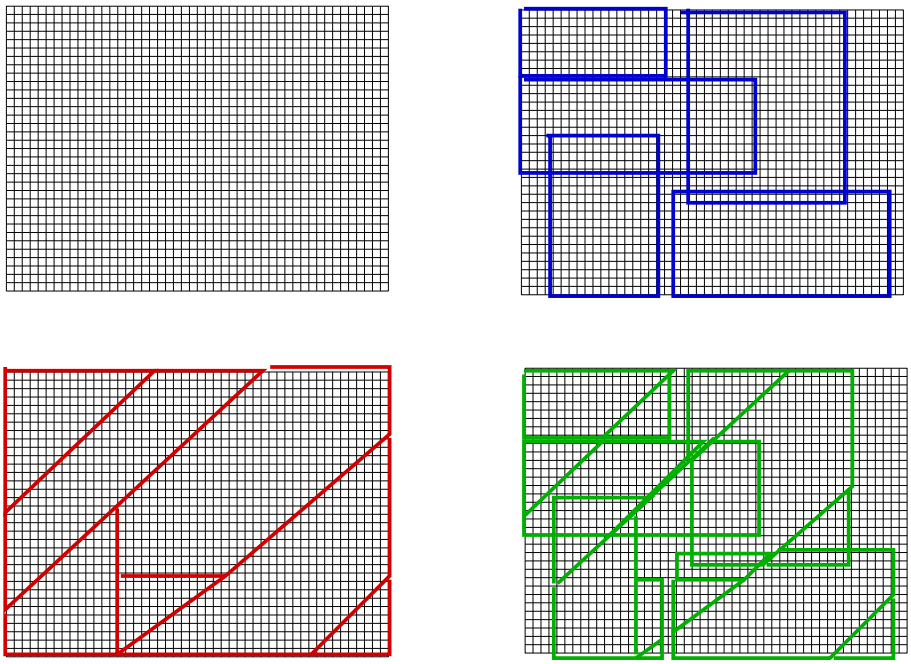


FIG. 8.4 – *La grille en haut à gauche représente l'espace d'états (chaque case correspond à un état). La grille en haut à droite représente le morcellement nul lié aux propriétés de la représentation intensionnelle du domaine. Chaque ensemble y correspond à l'ensemble d'états contenant une propriété de la modélisation. La grille en bas à gauche présente un morcellement lié à l'utilité des états. Finalement, pour planifier en optimisant, le morcellement de la grille en bas à droite est plus adapté. Certes, les regions délimitées sont plus petites, regroupant moins d'états, cependant elles restent plus grandes que la discrétisation en états individuels.*

Deux cas particulier doivent être évoqués :

1. Il est possible que chaque état individuel ait une valeur d'utilité différente. Nous retrouvons alors le morcellement total.
2. Au contraire, l'utilité peut être attachée aux caractéristiques individuelles du domaine de planification (c'est à ce cas qu'est forcé de se restreindre `DT-Graphplan`). Dans ce cas, le morcellement nul est adapté.

Dans ces deux situations, les morcellements respectifs de la planification et de l'optimisation forment les exacts mêmes ensembles d'états.

Le morcellement présenté dans cette section ne constitue pas une arborescence. Il définit

des zones d'utilité au travers desquelles cheminent les plans. La valeur des états, qui ne peut pas être encodée par la représentation en intension, l'est au travers du morcellement.

Morcellement, monotonie et *branch and bound*

Si l'utilité évolue de manière monotone avec l'exécution d'actions, ce morcellement peut être utilisé en filtrage. En effet, à partir du moment où un plan a déjà été évalué, seules les classes "meilleures" que sa valeur restent dignes d'intérêt. Pour la construction d'autres plans, il est alors inutile d'introduire des actions menant vers d'autres classes que celles qui sont "meilleures" que la valeur courante. Il s'agit d'une contrainte de classe imposée dynamiquement aux opérateurs du domaine de planification. La valeur du premier plan trouvé tient lieu de borne qui est exploitée pour éliminer certaines zones de l'espace de recherche définies par le morcellement. En cela c'est une application de l'approche *Branch and Bound* (cf. annexe A).

Dans la présentation de l'une de nos expérimentations de `TokenPlan` - celle concernant l'optimisation des déplacements d'avions au sol (cf. section 9.3), nous détaillons l'impact de ce type de filtrage sur la taille du graphe de planification.

En outre, ce filtrage permet d'accélérer la saturation du graphe de planification. C'est une sorte de "saturation pertinente" : nous ne nous intéressons qu'à la saturation du sous-graphe qui est "meilleur" que les plans qui ont déjà été trouvés.

Morcellement et abstraction

Ce morcellement peut aussi être utilisé en abstraction. Dans ce cas, le planificateur ne considère plus les états individuels, mais les zones définies par le morcellement intermédaire. La construction du graphe de planification permet d'établir les transitions entre ces zones. Les tables de transition de classe mémorisées à chaque niveau pour permettre de déployer les actions *NoOp* pertinentes pourraient être exploitées en ce sens. Cela revient à établir $\mathcal{F}_{branch}$ à partir de $\mathcal{F}_{morc}$ en excluant de plus les transitions intra-classes. Une nouvelle tâche de planification doit être menée pour extraire un plan en termes de ces transitions entre zones d'utilité.

8.2.3 Recherche arrière adaptée à un morcellement non arborescent

Les stratégies de morcellement que nous venons d'évoquer n'introduisent pas d'arborescence dans la structure du graphe de planification. Dès lors, la recherche d'un plan dans le graphe de planification ne revient pas à remonter une branche de sa feuille jusqu'à la racine.

D'ailleurs, les classes ne servent plus à cloisonner des ensembles de propositions, mais à apporter des informations supplémentaires. Imposer lors de la recherche arrière que toutes les propositions d'un niveau du plan appartiennent à une même classe n'a plus alors aucun sens.

Pour cette raison nous avons conçu pour `Tokenplan` une autre recherche arrière que celle que nous avons déjà présentée dans le chapitre précédent (cf. section 7.5).

Spécificité de cette recherche arrière

La seule différence entre les deux recherches arrières est qu'il n'est plus obligatoire que tous les jetons d'un même niveau soient dans une classe commune. Il est seulement nécessaire que ceux qui marquent les préconditions d'une même action soient dans une même classe.

La première conséquence de ce changement est que les objectifs peuvent être supposés atteints dans des classes différentes. Dans un tel cas, la recherche précédente retournerait un échec, alors que celle-ci poursuit son cours.

Condition de déclenchement de la recherche arrière

Si tous les objectifs sont marqués par des jetons non mutuellement exclusifs, **quelles que soient leurs classes respectives**, alors la construction du graphe de planification est interrompue, et une recherche arrière de plan est lancée.

De la même manière, au sein de chaque niveau les diverses propositions à soutenir peuvent l'être dans des classes différentes - sauf si elles sont préconditions d'une même action. Ainsi, il peut apparaître des sortes de recherches parallèles qui ne se coordonnent que lorsqu'elles soutiennent à plusieurs les différentes préconditions d'une même action.

Soulignons dès à présent que ce parallélisme apparent n'implique pas que ces recherches s'accomplissent en indépendance les unes des autres. En effet, les relations de mutuelle exclusion sont toujours prises en compte sur l'intégralité de chaque niveau.

Comme précédemment, un contrôle en avant du plan dégagé est nécessaire pour préciser les classes effectivement concernées. L'algorithme 3 (page suivante) détaille le choix des actions de support au sein d'un niveau. Par rapport à l'algorithme précédent, la structure des paramètres en entrée est un peu plus complexe. En effet, à chaque niveau, l'ensemble des buts à soutenir est structuré en sous-ensembles associés à des contraintes de classes différentes. Ainsi, au lieu d'une liste de buts et d'une liste de classes, l'algorithme prend comme paramètre une liste de couples ($\{buts\}$,$\{classes\}$). Bien-sûr, lors du lancement de la recherche arrière, l'ensemble de buts de chaque couple contient un unique but. L'ensemble de classes de chaque couple ($\{buts\}$,$\{classes\}$) rassemble les classes dans lesquelles l'ensemble de buts du couple apparaît au niveau considéré, non mutuellement exclusif avec les autres buts.

Ce nouvel algorithme met en œuvre deux sous-programmes. Le second est similaire à l'algorithme de la première recherche. Ici, cependant, il n'est appelé que pour traiter un sous-ensemble de buts avec ses contraintes de classes. Le premier sous-programme a, lui, la charge de l'intégralité d'un niveau. Il appelle le second sous-programme avec chaque sous-ensemble de buts et accumule au fur et à mesure les actions sélectionnées. Cette accumulation progressive est importante. Elle permet en effet de contrôler les relations de mutuelle exclusion sur l'intégralité du niveau.

La figure 8.5 reprend l'exemple du chapitre précédent (Fig. 7.11) pour le traiter avec cette nouvelle recherche.

Algorithme 3: Pendant la recherche arrière il faut tenir compte des classes pour en tirer parti. Cet algorithme prend en charge l'assignation d'actions à une série de buts au sein d'un même niveau. L'appel initial se fait avec une liste de couples ($\{buts\}$,$\{classes\}$), et deux ensembles vides pour les actions déjà sélectionnées, et les prochains ensembles de buts avec leurs classes associées (pour le niveau traité ensuite).

AttribuerSupportNiveau(BetCsets, actions, next-BetCsets)
si $BetCsets = \emptyset$ **alors**
  retourner *actions, next-BetCsets*
sinon
  $(buts, classes) \leftarrow depiler(BetCsets)$
  $(actions2, next-BetCsets2) \leftarrow AttribuerSupportButs(buts,classes,actions,\emptyset,\emptyset)$
  si *actions2* $\neq \emptyset$ **alors**
    AttribuerSupportNiveau(BetCsets, actions2, next-BetCsets + next-BetCsets2)
  sinon
    Echec

AttribuerSupportButs(Buts, Classes, actions, nextButs, nextClasses)
si $Buts = \emptyset$ **alors**
  retourner *actions, (nextButs, NextClasses)*
sinon
  $but \leftarrow depiler(Buts)$
  pour *chaque action dans support(but)* **faire**
    classes-prec-admissibles$\leftarrow \{c \in classes(prec(action))/classes(effets(actiondansc)) \cap Classes \neq \emptyset\}$
    si *classes-prec-admissibles* $\neq \emptyset$ *et* $\neg$ *mutex(action, actions)* **alors**
      AttribuerSupport(Buts, Classes, actions $\cup$ {action}, nextButs $\cup$ prec(action), nextClasses $\cap$ classes-prec-admissibles)

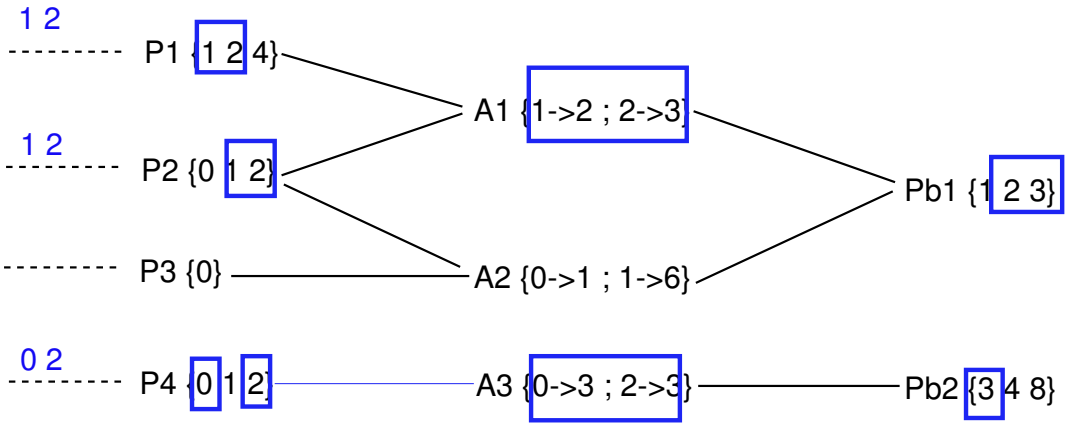


FIG. 8.5 – *Le même exemple que celui de la figure 7.11 est repris ici. Cette fois, il n'est pas nécessaire que les buts soient dans une classe commune. Ainsi, Pb*1 *peut être soutenue dans les classes* 1 2 *ou* 3 *et Pb*2 *dans les classes* 3 4 *ou* 8*. Considérons Pb*1*. Avec ce mode de recherche A*1 *comme A*2 *peuvent être retenues. Supposons que A*1 *le soit. Pb*1 *est alors supportée dans les classes* 2 *et* 3*. Il faut pour cela que les préconditions soient soutenues dans les classes* 1 *ou* 2*. De manière analogue, Pb*2 *peut être soutenue dans la classe* 3 *par A*3*. Supposons pour l'exemple que les préconditions de A*1 *et A*3 *ne sont pas mutuellement exclusives. La recherche peut donc se poursuivre. P*1 *et P*2 *doivent être soutenues dans la classe* 1 *ou* 2*, mais toutes deux dans la même. P*4 *doit être soutenue dans la classe* 0 *ou* 2*.*

8.3 Discussion sur les deux variantes de recherche

La première recherche est la plus contrainte des deux. Elle force chaque étape du plan à se contenir dans une seule classe - donc dans un seul morceau de l'espace. Le morcellement est utilisé pour distinguer les choses, pour veiller à ce qu'il n'y ait pas de mélange.

La deuxième variante, elle, exploite le morcellement pour définir des contributions. Typiquement, plusieurs classes peuvent contribuer à l'état final.

Suivant le problème traité, et la stratégie de morcellement mise en œuvre, l'une comme l'autre sont plus ou moins adaptées.

D'un point de vue plus technique, ces deux variantes se distinguent aussi au niveau du comportement même de la recherche de plan. La première variante étant plus contrainte, elle rencontre plus rapidement des échecs. Ainsi, lors de chaque recherche elle produit moins de retours arrières. En revanche, elle nécessite de développer le graphe plus loin.

Il s'ensuit par ailleurs que la seconde variante peut produire des plans comportant plus d'actions en parallèle que la première. En effet, elle autorise plusieurs actions au sein d'un même niveau même si elles ne correspondent pas aux mêmes transitions de classes. Ce n'est pas le cas de la première variante. Cette différence apparaît clairement dans les résultats de l'expérimentation concernant les déplacements d'avions au sol (cf. section 9.3 dans le chapitre suivant).

Chapitre 9

Expérimentations

`TokenPlan` a été implémenté en *Common Lisp*. Cette implémentation est dérivée d'une implémentation de `Graphplan` de Kambhampati et al qui intègre ses propres travaux sur l'utilisation des techniques issues de CSP pour la recherche arrière.

Le logiciel obtenu n'est pas optimisé. Il s'agit d'un outil nous permettant de valider notre point de vue sur le morcellement de l'espace de recherche et son potentiel en planification d'actions.

La première expérimentation concerne l'utilisation du morcellement de l'espace de recherche pour encoder une partie des relations de mutuelle exclusion. Cette réflexion nous a mené à développer un algorithme spécifique intégré dans la mécanique même de `TokenPlan` pour le calcul des relations de mutuelle exclusion.

Ensuite, deux types d'expérimentations sont présentées. Le premier vise à montrer que `TokenPlan` peut utiliser un morcellement nul, comme `Graphplan`. Ces expérimentations valident l'héritage de la planification classique disjonctive que porte `TokenPlan`.

Le second type d'expérimentations exploite des morcellements intermédiaires pour traiter des problèmes d'optimisation, qu'il s'agisse d'optimisation de durée ou de critères plus courants. En particulier, les dernières expérimentations montrent que l'exploitation d'un morcellement intermédiaire adapté permet à `TokenPlan` de traiter les problèmes habituellement traités par la programmation dynamique.

9.1 Une exploitation particulière du morcellement pour encoder des relations de mutuelle exclusion

Nous l'avons évoqué à plusieurs reprises : le morcellement peut être exploité pour encoder des relations de mutuelle exclusion. Nous revenons sur ce type d'exploitation pour encoder plus particulièrement les couples mutuellement exclusifs.

Nous montrons qu'une description différente des classes est plus appropriée à cet encodage des couples mutuellement exclusifs. Enfin, ayant décidé d'intégrer ce type d'approches au module de `TokenPlan` en charge du calcul des relations de mutuelle exclusion, nous avons construit un algorithme approprié. Ainsi, `TokenPlan` utilise toujours cette approche pour une partie de la détection des relations de mutuelle exclusion, indépendemment des stratégies de morcellement par ailleurs mises en œuvre.

9.1.1 Un morcellement adapté peut encoder les relations binaires de mutuelle exclusion

Deux propositions mutuellement exclusives ne doivent pas être considérées ensemble par le planificateur lorsqu'il construit un plan. Une manière d'assurer le respect de cette contrainte est d'utiliser un morcellement arborescent qui sépare les couples mutuellement exclusifs.

Chaque classe ne contient que des propositions non mutuellement exclusives entre elles. Le choix d'un morcellement arborescent permet de ne pas mélanger ces classes - chacune étant dans une branche différente. Les couples de propositions interdits ne peuvent pas ainsi être considérés.

La figure 9.1 montre un exemple de début de graphe de planification. Le deuxième niveau propositionnel contient quatre propositions : $P3$, $P4$, $P5$ et $P6$. Cet ensemble contient deux couples mutellement exclusifs : $P3$ est mutellement exclusive avec $P4$, et $P5$ avec $P6$. Pour les prendre en compte, quatre classes doivent être créées définissant les quatre ensembles suivants : $\{P3, P5\}$, $\{P4, P5\}$, $\{P3, P6\}$ et $\{P4, P6\}$. $P3$ et $P4$ ne sont jamais ensemble, ni $P5$ et $P6$. En revanche, toutes les combinaisons possibles sont conservées.

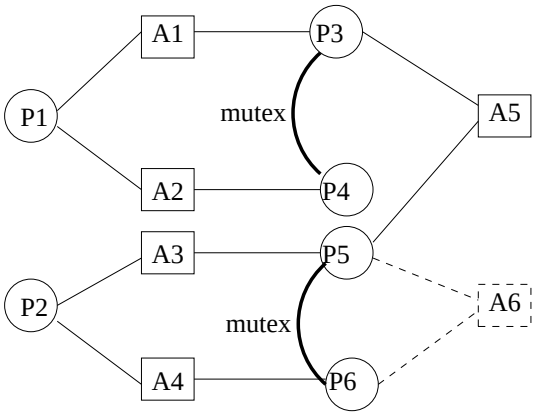


FIG. 9.1 – *Début d'un graphe de planification. Le deuxième niveau propositionnel contient deux couples mutuellement exclusifs, de sorte qu'au niveau suivant l'action $A6$ ne peut pas être introduite.*

Remarquons que sur cet exemple un morcellement total aurait aussi créé 4 classes. Cependant, chacune d'entre elles aurait été associée à une seule action. Ici, l'éventuelle exécution parallèle des actions est prise en compte. Les propositions de la deuxième classe, par exemple, sont obtenues en exécutant à la fois les actions $A2$ et $A3$.

9.1.2 Exploiter son "négatif" plutôt que le morcellement lui-même

L'espace des ensembles de propositions non mutuellement exclusives est très morcelé dans les domaines de planification courants. Il y a deux raisons à cela :

1. d'une part, beaucoup de propositions différentes sont concernées par des relations de mutuelle exclusion ;
2. d'autre part, ces propositions ne sont pas mutuellement exclusives avec beaucoup d'autres propositions.

De fait, lorsqu'un nouveau couple mutuellement exclusif ($P1$,$P2$) apparaît, beaucoup des classes existantes sont concernées, mais peu d'entre elles contiennent uniquement des éléments mutuellement exclusifs. En conséquence, toutes ces classes doivent être dédoublées :

- d'un côté les propositions de la classe considérée compatibles avec $P1$;
- de l'autre côté les propositions de la classe considérée compatibles avec $P2$.

Ainsi, un seul nouveau couple interdit peut générer un grand nombre d'ensembles admissibles à considérer.

Il apparaît donc plus raisonnable de considérer les interdictions plutôt que les autorisations. Cela revient à ne pas décrire les classes elles-mêmes mais ce qui les définit, ce qui les distingue les unes des autres.

Il ne s'agit plus de réunir les propositions qui peuvent être ensemble - comme le font les classes - mais de marquer celles qui ne doivent pas être ensemble. Pour ce faire et pour distinguer ces ensembles particuliers de classes, nous exploitons la notion de *couleur des jetons* qui existe dans le cadre des réseaux de Petri.

La contrainte apportée par les couleurs au déclenchement des transitions est contraire à celle des classes. C'est parce qu'elles représentent des interdictions au lieu d'autorisations.

Règle : couleurs et relations de mutuelle exclusion

Deux jetons de même couleur sont mutuellement exclusifs. Une transition dont deux de ses préconditions sont marquées par des jetons de même couleur ne peut donc pas être déclenchée.

Comme c'est le cas avec les classes, un jeton peut avoir plusieurs couleurs. Ces différentes couleurs le rendent mutuellement exclusif avec plusieurs ensembles de jetons.

9.1.3 Coupler les couleurs à la dynamique des jetons

C'est pour gagner en performances, dès la construction du graphe, que nous encodons les relations de mutuelle exclusion par un morcellement spécifique. Dans ce but, nous avons choisi de ne pas calculer les relations de mutuelle exclusion par les méthodes traditionnelles pour déterminer les classes. Au contraire nous nous appuyons le plus possible sur le morcellement des niveaux précédents pour déterminer le morcellement du niveau courant.

L'utilisation d'un réseau de Petri par `Tokenplan` permet d'exploiter les propriétés physiques des jetons. Normalement, Un jeton ne peut pas passer au travers de deux transitions en même temps. C'est aussi le cas pour les ressources non partageables et des agents d'un domaine. Ce sont eux qui causent les relations permanentes de mutuelle exclusion.

Les couleurs que nous utilisons pour coder de manière compacte un morcellement de mutuelle exclusion est adapté à la description de ce type de comportement. Attribuons à chaque ressource non partageable, et à chaque agent, une couleur spécifique. C'est cohérent avec la sémantique des ensembles définis par les couleurs car chaque ressource non partageable ou agent est mutuellement exclusif avec lui-même : en effet, il n'est pas possible d'en utiliser deux instances en même temps.

La propagation du morcellement suit alors la propagation des couleurs. Par exemple, soient $A1$ et $A2$ deux actions qui consomment une ressource non partageable colorée en bleu. $A1$ et

$A2$ sont mutuellement exclusives (par la *règle d'interférence*). Les deux effets obtenus via ces actions par la consommation de la ressource non partageable sont évidemment mutuellement exclusifs. S'ils sont restés bleus, en héritant de la couleur de la ressource mère, alors ils sont automatiquement codés mutuellement exclusifs, sans besoin d'autres calculs.

Intuitivement, l'idée est d'utiliser la non-ubiquité intrinsèque des jetons pour encoder le fait que certaines ressources, et les agents du domaine, ne sont pas partageables. La couleur nous permet de tracer la destinée d'un jeton. C'est comme introduire du colorant dans une source puis s'en servir pour repérer l'interconnexion des divers cours d'eau.

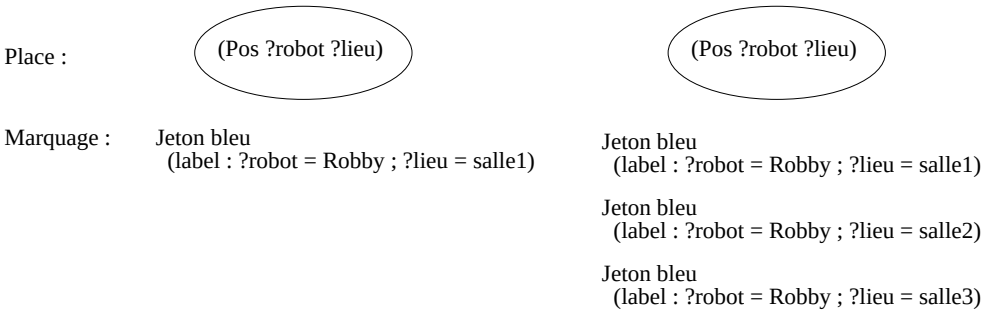


FIG. 9.2 – *Le robot Robby se trouve initialement dans la pièce* 1. *Un jeton bleu portant le label correspondant marque la place Pos (pos pour position). Robby peut rester où il est ou bien se rendre dans les pièces* 2 *et* 3. *Au niveau suivant, la même place est donc marquée par trois jetons, avec des labels différents. Ces trois jetons sont tous bleus puisqu'ils descendent du même jeton initial - lui-même bleu. Ils s'ensuit qu'ils sont mutuellement exclusifs - d'après la règle sur les couleurs des jetons. Comme le montre la figure 4.6, dans le même cas et sans les couleurs, trois relations* mutex *traditionnelles devraient être explicitées*

La figure 9.2 reprend l'exemple utilisé pour présenter les relations permanentes de mutuelle exclusion (fig. 4.6 du chapitre 4 de la première partie). Toutes les relations *mutex* permanentes peuvent y être encodées au travers des couleurs. La figure 9.3 montre un exemple similaire avec deux robots. Toutes les mutuelles exclusions (toutes permanentes) sont bien détectables grâce aux couleurs, et il n'y a pas de relation de mutuelle exclusion erronée.

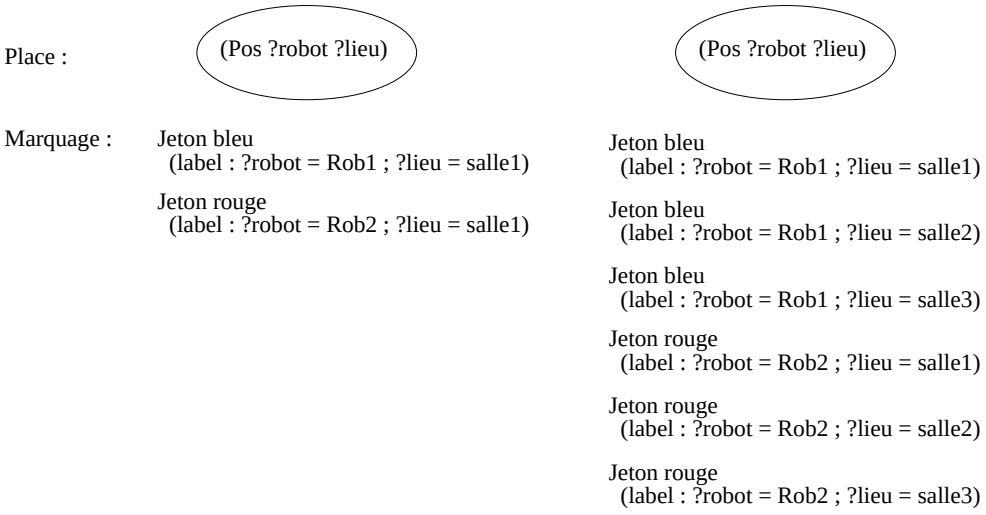


FIG. 9.3 – *Dans cet exemple, deux robots sont considérés au lieu d'un seul. Dans l'état initial, chaque jeton a une couleur différente (ici bleu ou rouge). Au niveau suivant chaque robot peut être dans l'une des trois salles* 1, 2 *et* 3. *Les couleurs encodent bien le fait qu'un robot ne peut pas être dans deux pièces différentes, sans empêcher deux robots d'être dans une même pièce, car une pièce est ici une ressource partageable.*

9.1.4 Intégration pratique au sein de `TokenPlan`

Dans la généralité d'une description de domaine de planification, les notions *d'agents* et de *ressources* - partageables ou non - ne sont pas définies. Seule existe la notion de *propriété*.

Un seul agent peut être l'objet de plusieurs propriétés - décrivant plusieurs aspects dudit objet. Une seule propriété peut concerner plusieurs agents ou ressources non partageables. Ainsi, un jeton ne représente pas une ressource ou un agent, mais une propriété seulement.

Par ailleurs, si un jeton participe au déclenchement d'une action ayant plusieurs effets, ces effets ne sont pas mutuellement exclusifs entre eux. La propagation des couleurs que nous avons présentée laisse pourtant penser le contraire.

Pour ces raisons, l'exploitation des couleurs et leur propagation telles que nous les avons intégrées à `TokenPlan` sont un peu différentes. Tout d'abord, la relation entre les couleurs et les mutuelles exclusions est plus faible.

Règle : couleurs et relations de mutuelle exclusion au sein de `TokenPlan`

Deux jetons de même couleur, ***marquant une même place avec des labels différents*** *sont mutuellement exclusifs.*

Ensuite, la propagation des couleurs est adaptée au mode de description "à la STRIPS" d'un domaine de planification. `Tokenplan` utilise pour le marquage initial des jetons qui ont tous une couleur différente. Il est supposé en effet que le niveau initial ne contient pas de relation de mutuelle exclusion. Si cette condition n'est pas remplie, cette mthode ne peut pas tre utilisée. Nous pensons que cette restriction n'est pas une contrainte forte car ce niveau décrit généralement **l'état** initial.

L'intégrité des relations de mutuelle exclusion ne peut être maintenue que si les couleurs se propagent en même temps que les jetons de manière adéquate. Par exemple, comment répartir les couleurs des jetons marquant les places préconditions d'une transition parmi les jetons marquant ses effets? Si le nombre d'effets, n'est pas le même que celui des préconditions, faut-il supprimer ou créer des couleurs? Comment discerner les ressources partageables de celles qui ne le sont pas?

Il serait préférable de pouvoir attribuer une couleur à chaque objet ou ressource non partageable, puis de maintenir la cohérence entre couleurs et objets au fil des transitions. Au lieu de cela `TokenPlan` s'attache à maintenir une cohérence entre couleurs et certaines propositions : celles de l'état initial.

Cette approche n'est pas aussi précise. Cependant, elle assure tout de même de ne pas introduire de relation de mutuelle exclusion qui n'aurait pas lieu d'être. En particulier, si le marquage de l'état initial réapparait plus loin dans la construction de l'espace de recherche, cette approche nous permet d'assurer que la coloration des jetons soit identique dans les deux cas.

Par ailleurs, les différents objets sont introduits par l'état initial. En attribuant une couleur à chaque jeton, nous attribuons par la même occasion une couleur à chaque objet (donc à chaque agent, et à chaque ressource non partageable). En revanche, lorsqu'un objet apparait dans plusieurs propositions, il apparaît sous plusieurs couleurs. Pour cette raison des relations de mutuelle exclusion peuvent nous échapper, mais aucune ne peut être détectée par erreur.

Voici comment le détail de la propagation des couleurs au travers d'une transition. La figure 9.4 présente un exemple complet.

Lorsqu'une transition est déclenchée, le jeton de transition porte la couleur C_{action}. C_{action} est l'union des couleurs des jetons marquant les préconditions. Les jetons marquant une place précondition qui est aussi un effet implicite de l'action considérée ne sont pas pris en compte. Ceci écarte les ressources partageables.

A chaque place est associée la liste des couleurs des jetons qui la marquaient dans le marquage initial. Les couleurs des effets sont calculées à partir de C_{action} et de ces couleurs associées aux places des effets.

Les places-effets qui ont déjà été marquées dans le marquage initial, ne gardent de C_{action} que les couleurs mémorisées. Ce mécanisme permet de retrouver les couleurs originelles à partir des ensembles de couleurs qui ont été construits par les transitions successives.

Ces couleurs sont retirées de C_{action}. L'ordre de traitement de ces effets n'a pas d'importance puisque tous les jetons avaient des couleurs différentes dans l'état initial. Les places effets qui n'était pas marquées initialement, reçoivent toutes le reste de C_{action}.

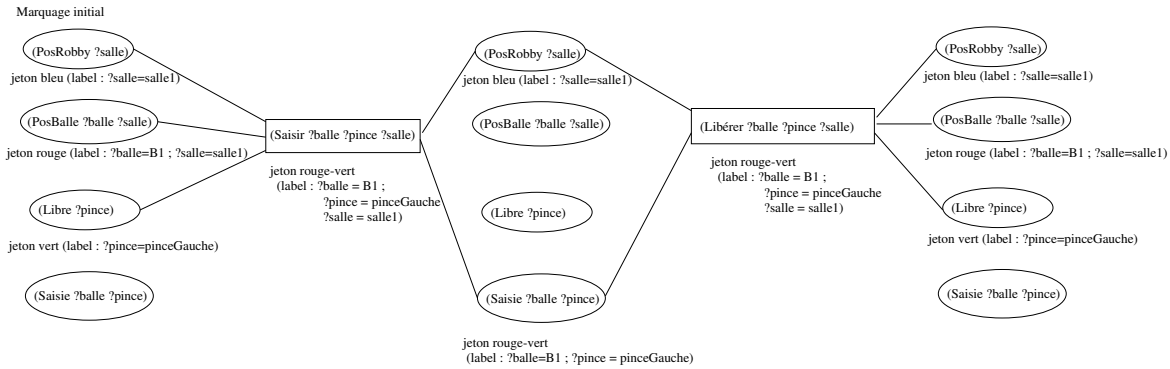


FIG. 9.4 – *Voici un enchaînement de deux actions à partir de l'état initial. Ces deux actions sont symétriques : la seconde défait ce que la première a fait. En conséquence, les marquages du niveau initial et du troisième niveau de places sont identiques. Nous constatons que les couleurs des jetons sont aussi identiques.*
Dans le détail, le jeton de la première transition porte les couleurs rouge et vert. La couleur bleue n'est pas retenue car elle correspond à un jeton lié à un effet implicite, c'est-à-dire un jeton utilisé mais pas consommé. Le seul effet non implicite n'était pas marqué dans l'état initial, il reçoit donc l'intégralité de la couleur du jeton de transition.
Lors de la deuxième transition, la couleur du jeton est encore une fois rouge-vert. Le jeton bleu suit encore une fois un lien vers un effet implicite. La place PosBalle était marquée initialement par un jeton rouge. Elle récupère donc la couleur $\{rouge, vert\} \cap \{rouge\}$ *- soit la couleur rouge. Pour des raisons similaires, la place Libre est marquée par un jeton vert.*

9.1.5 Considérations d'implémentation et de performances

Pour représenter les couleurs, `TokenPlan` utilise des entiers. Le test de mutuelle exclusion se limite à une simple intersection qu'il serait facile d'optimiser à l'aide d'opérations bit à bit (cette optimisation n'est pas présente dans l'implémentation actuelle de `TokenPlan`).

En effet, l'intérêt d'encoder une partie des relations de mutuelle exclusion par les couleurs est de pouvoir les déceler plus facilement. Bien-sûr, si deux jetons n'ont pas de couleurs mutuellement exclusives, il est vérifié s'ils sont quand même mutuellement exclusifs d'après les règles d'interférence, de compétition des ressources pour les jetons de transition, ou en analysant leurs supports dans le cas de jetons marquant des propositions. Cette vérification garantit que toutes les relations de mutuelle exclusion sont prises en compte. La figure 9.5 montre un exemple où l'utilisation des couleurs joue un rôle important en termes de performances.

	Graphplan			TokenPlan		
	64 s 550 msec			4s 800 msec		
level	prop mut	action mut	records	prop mut	action mut	records
0	214	3858	0	136	2478	0
1	214	3858	1	136	2478	1
2	214	3858	45	136	2478	46
3	214	3858	254	136	2478	103
4	214	3858	1087	136	2478	204
5	214	3894	2724	136	2478	297
6	226	4206	5746	136	2478	781
7	250	4182	10774	136	2604	1261
8	322	3298	10237	148	1458	631
9	542	1466	0	112	680	0
10	220	176	0	140	121	0
11	0	0	0	0	0	0

FIG. 9.5 – *Comparaison sur le problème du* GRIPPER *à 6 balles, entre* **TokenPlan** *en morcellement nul utilisant les couleurs mutuellement exclusives, et une version de* **Graphplan** *(1999) fournie par S. Kambhampati. Seules les relations* mutex *calculées sans les couleurs sont prises en compte dans ce tableau. La différence entre* **TokenPlan** *et* **Graphplan** *provient des relations* mutex *permanentes déduites des couleurs.*

Sur cet exemple, entre 36% et 84% des relations de mutuelle exclusion sont encodées par les couleurs. Le gain en temps de résolution est impressionnant : de l'ordre de 13 fois plus rapide.

Les avantages en termes de performances de cette utilisation des couleurs sont variables suivant les problèmes traités. En particulier, l'écriture des domaines a une grande influence sur la distinction entre les ressources et les objets. Ces résultats sont en accord avec ceux d'autres travaux visant à exploiter le caractère invariant de certaines relations de mutuelle exclusion([24], [47]).

L'exploitation de l'encodage de tout ou partie des relations de mutuelle exclusion pour améliorer les performances du planificateur reste une voie prometteuse. Plusieurs travaux y consacrent leurs efforts (par exemple [76], [32]).

9.2 Morcellement nul : émulation de `Graphplan`

`TokenPlan` a participé à la compétition de planification de la conférence *AIPS* en 2000. Cette participation nous a permis de valider notre cadre à base de réseaux de Petri et de classes dans le cadre du morcellement nul [55]. En effet, l'objet de `TokenPlan` étant de permettre l'expérimentation de nombreuses stratégies de morcellement, il fallait nous assurer qu'il reproduisait convenablement le comportement des planificateurs utilisant des stratégies déjà connues.

Compte tenu des différences de langages d'implémentation (`TokenPlan` est implémenté en Lisp, d'autres planificateurs étaient écrits en C ou en C++), les résultats montrent que `TokenPlan` est dans le flot des planificateurs liés directement ou indirectement à `Graphplan` (cf. Fig. 9.6). Ses performances sont un peu meilleures que `BlackBox`, bien que ce dernier s'en sorte mieux face à de gros problèmes, et elles sont à peu près équivalentes à celles de `IPP` [55]. Ces deux planificateurs sont des dérivés de `Graphplan`.

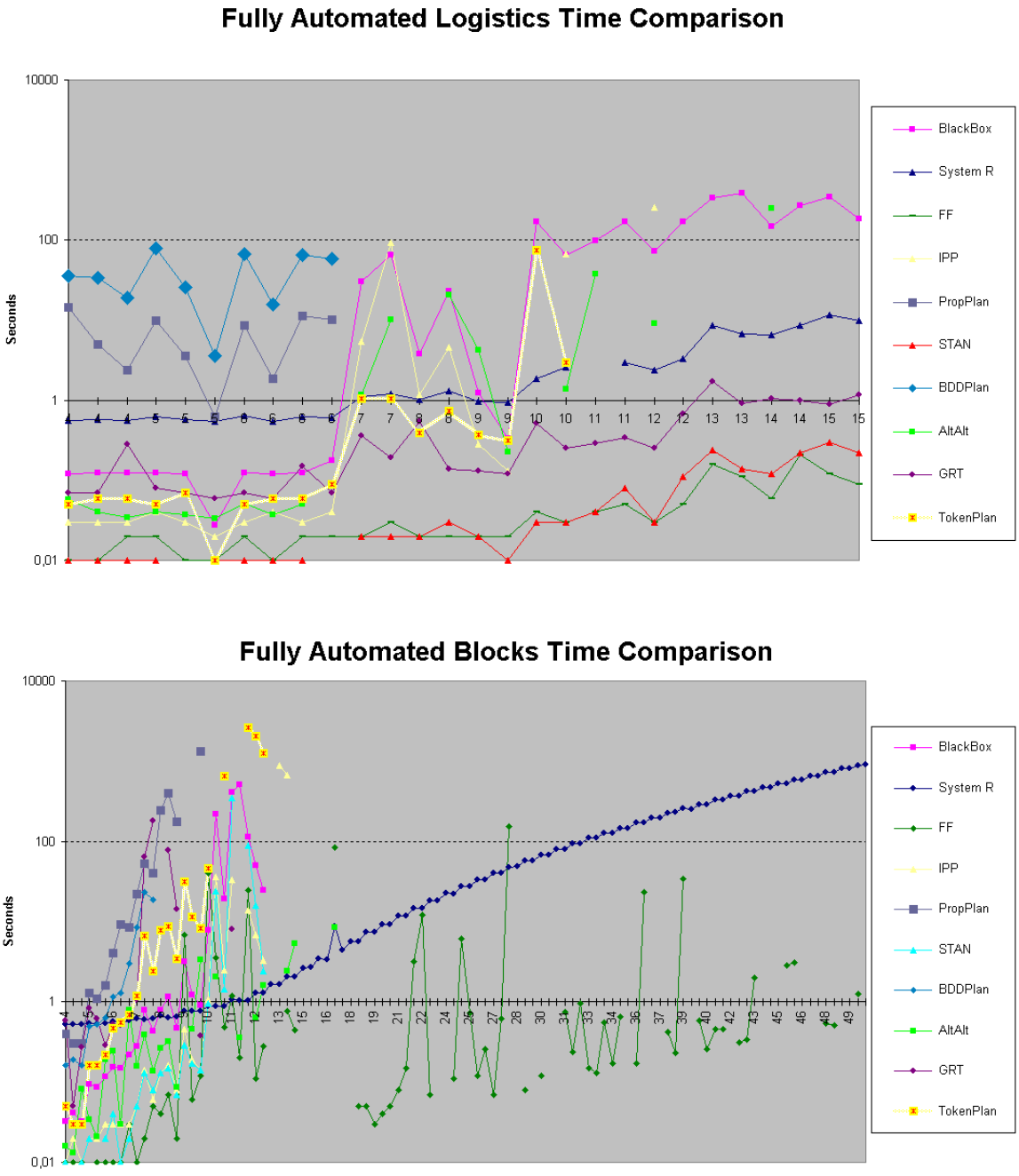


FIG. 9.6 – *Ces deux figures montrent les performance en temps CPU de quelques uns des planificateurs en compétition à AIPS'2000. Les points manquant correspondent à des problèmes qui n'ont pas pu être résolus dans la limite de temps-CPU imposée (30 minutes). Le graphe du haut montre les résultats sur le domaine* Logistics *et celui du bas sur le domaines* des blocs. *Dans ce dernier cas l'abscisse correspond au nombre de blocs considérés. Dans un cas comme dans l'autre la présentation sous forme de courbe permet de suivre plus facilement le comportement de chaque planificateur. Cependant il ne faut supposer aucun ordre rigoureux quant à l'attribution d'une abscisse à un problème. Tous les planificateurs ont tourné sur un Pentium III à 500MHz avec 1GB de RAM. Ces données nous ont été communiquées par F. Bacchus.*

9.3 Morcellement intermédiaire : planification de mouvements au sol d'avions dans un aéroport

9.3.1 Un problème d'optimisation

Le problème à traiter est celui de la gestion des déplacements au sol des avions sur un aéroport. Le nombre de ces déplacements augmente avec le trafic aérien. Les fortes contraintes qui s'appliquent à ces déplacements leurs confèrent une place importante dans la gestion globale du trafic aérien, et ils pourraient devenir l'une des principales sources des retards aériens.

Ces contraintes sont liées aux aéroports et aux avions eux-mêmes. D'une part, l'espace utilisable au sol est réduit. Les avions sont le plus souvent obligés de se suivre sur une voie unique. Le moindre incident concernant le déplacement d'un avion bloque ainsi tout le reste de la file. D'autre part, les avions sont peu manœuvrables à terre : pas de marche arrière, demi-tours difficiles et nécessitant l'intervention de matériel supplémentaire (camions de traction...).

Il ne s'agit pas ici de planifier l'intégralité des mouvements sur un aéroport, et d'optimiser l'allocation des portes d'embarquement. Notre objectif est de résoudre une situation locale, le plus souvent conflictuelle. Suite à un incident, ou à une erreur, un avion s'est engagé sur une voie qu'il n'était pas censé emprunter. Les procédures de déplacement suivies par plusieurs avions au sol à ce moment-là peuvent alors se retrouver inappropriées. Compte tenu du positionnement des avions, le planificateur doit générer un plan de déplacement pour la flotte qui permette à chaque avion de rejoindre la position qu'il doit atteindre. Le sol de l'aéroport retrouvera alors une configuration normale et les procédures de déplacement standards pourront reprendre. Les figures 9.7 et 9.8 montrent les configurations initiales et finales d'un exemple de problème à traiter.

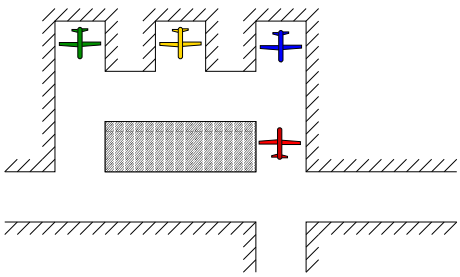


FIG. 9.7 – *Etat initial*

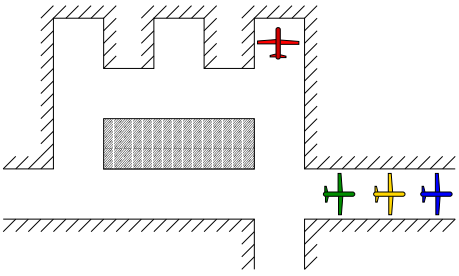


FIG. 9.8 – *Etat final souhaité*

Le critère à optimiser est le temps que mettra la flotte considérée pour retrouver la configuration souhaitée. C'est un critère particulier en ceci qu'il ne peut pas être recomposé additivement à partir des temps de déplacement de chaque avion. Si un avion A se déplace pendant 10 minutes et qu'un avion B se déplace simultanément pendant 10 minutes, alors les deux avions se sont déplacés globalement pendant 10 minutes, et non pas pendant 20 minutes.

9.3.2 Modélisation

La modélisation que nous avons choisie utilise les classes pour encoder le critère à optimiser. Ainsi, chaque classe correspond à une durée depuis le début de l'exécution du plan. Toutes les propositions de l'état initial sont portées par des jetons de la classe 0, puisque c'est l'instant initial. Les actions nécessitant toutes du temps pour s'exécuter, elles déclenchent des transitions vers les classes supérieures. Pour être conforme à la réalité, l'action de demi-tour réclamant plus de temps, elle cause un saut de 5 classes au lieu d'une seule.

Les taxiways et parkings de l'aéroport sont discrétisés en tronçons. Chaque tronçon est un objet du problème à traiter. La figure 9.9 représente l'aéroport et sa discrétisation utilisés dans nos exemples.

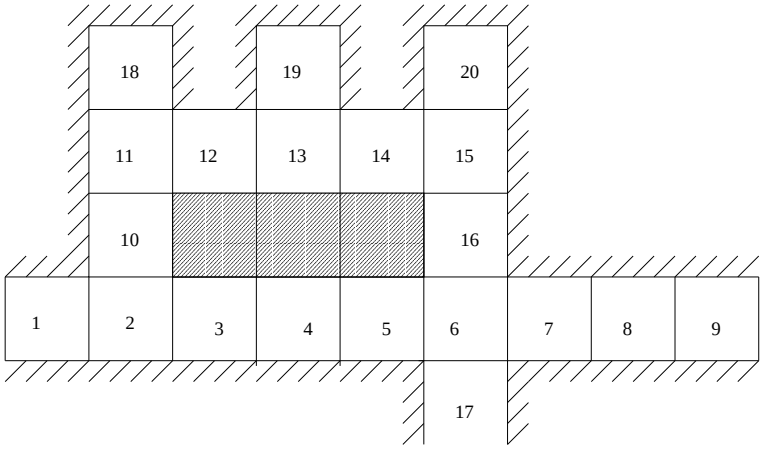


FIG. 9.9 – *Aéroport discrétisé en* 20 *tronçons*

Modélisation : topologie de l'aéroport

Deux prédicats permettent de décrire la topologie de l'aéroport :

- $(connected\ ?TR1\ ?TR2\ ?D)$. Les variables $?TR1$ et $?TR2$ désignent deux tronçons, et $?D$ une direction géographique ($Nord$, Sud, Est, $Ouest$). Ce prédicat indique qu'il est possible de passer du tronçon $TR1$ au tronçon $TR2$, en un coup (les tronçons $?TR1$ et $?TR2$ sont consécutifs), en se déplaçant dans la direction $?D$. Si les tronçons sont peuvent être utilisés dans les deux sens (c-à-d. s'ils ne sont pas en sens unique), il faut utiliser deux instances de ce prédicat, une pour chaque sens de circulation.
- $(free\ ?TR)$. Ce prédicat indique que le tronçon $?TR$ peut être occupé par un avion (il n'est pas déjà occupé, ni en travaux par exemple).

Modélisation : état d'un avion

Deux prédicats permettent de décrire l'état d'un avion :

- $(at\ ?P\ ?RP)$ indique que l'avion $?P$ se trouve sur le tronçon $?RP$
- $(oriented\ ?P\ ?D)$ indique que l'avion $?P$ est orienté dans la direction $?D$.

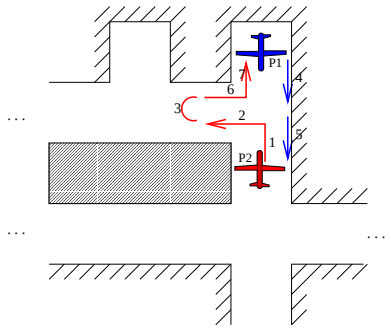


FIG. 9.10 – *Voici le plan obtenu avec le type de recherche le plus contraint. P1 attend que P2 ait terminé son demi-tour avant de démarrer. Il n'y a aucune justification réaliste pour cela.*

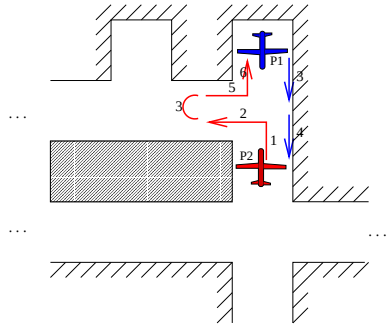


FIG. 9.11 – *Voici le plan obtenu avec le type de recherche le moins contraint. P1 démarre aussitôt que le taxi-way est disponible.*

Les actions de déplacement sont modélisées dans le référentiel local de chaque avion : avancer, tourner à gauche, à droite et faire demi-tour. Afin de tenir compte des contraintes d'orientation des avions sur les taxiways, et puisque cette version de `TokenPlan` ne gère pas les effets conditionnels, chaque possibilité de déplacement correspond à quatre opérateurs. Ainsi, il existe un opérateur *TournerGauche* qui a pour précondition d'être orienté vers le nord et pour effet d'être orienté vers l'ouest. Un autre a pour précondition d'être orienté vers l'est et pour effet d'être orienté vers le nord, et ainsi de suite pour chaque direction et pour chaque mouvement.

Chaque opérateur cause un changement de classe :

- Avancer et tourner à droite ou à gauche causent un saut d'une classe. Quelque soit la classe c des préconditions, les effets appartiennent à la classe $c + 1$;
- un demi-tour, du fait de sa difficulté, cause un saut de 5 classes.

9.3.3 Différences sensibles entre les deux recherches arrières

En utilisant la recherche la plus contrainte (la première que nous avons présentée, avec le morcellement arborescent cf. section 7.5), nous limitons le nombre de possibilités et donc de *backtracks*. Sur cet exemple, il s'ensuit qu'un plan solution est trouvé plus rapidement. Evidemment, ces contraintes se reflètent sur les plans trouvés. Ceux-ci sont plus "longs" car plus séquentiels. Par exemple, les actions $Avancer$ et $Demi - Tour$ ne peuvent jamais apparaître dans un même niveau parce qu'il est impossible d'avoir à la fois toutes leurs préconditions dans une même classe et tous leurs effets dans une même classe aussi. En effet, elles causent un "saut" d'une classe pour la première, et de cinq pour la seconde. Pourtant, intuitivement ces deux actions pourraient très bien démarrer en même temps (cf. fig. 9.10 et 9.11).

L'autre recherche arrière évite cet écueil. C'est parce qu'elle n'impose pas une classe unique par niveau (cf. section 8.2.3).

9.3.4 Impact du filtrage des classes à la "branch and bound"

Ce problème de déplacement d'avions au sol est un problème d'optimisation. Nous voulons minimiser la durée de déplacement de la flotte d'avions concernée. `Tokenplan` ne se contente pas de s'arrêter dès le premier plan trouvé. Il recherche le meilleur plan.

Le critère à minimiser est monotone : soit d la durée d'un plan partiel P ; en ajoutant une action A à P, nous obtenons un plan partiel $P + A$ dont la durée ne peut pas être inférieure à d. Ainsi d est une borne inférieure de la durée du meilleur plan qui pourra être construit à partir de P. Cette caractéristique permet d'exploiter les fonctionnalités de type *Branch and bound* de `Tokenplan`.

Trivialement, la partie "séparation" (*branch*) de l'espace de recherche est faite au travers du morcellement. Comme il se trouve que chaque classe correspond à une valeur du critère à minimiser, et que cette valeur est une borne inférieure du coût des plans qui pourront être obtenus avec les jetons de cette classe, "l'évaluation" (*bound*) est facile à mettre en œuvre.

Dès qu'un premier plan est trouvé, son coût est utilisé comme borne supérieure b_{sup} du coût du meilleur plan. Deux types de filtrages sont alors opérés par `Tokenplan`. D'une part, tous les nœuds du dernier niveau développé appartenant à des classes supérieures ou égales à b_{sup} ne sont pas reportés par les actions *NoOp*. C'est équivalent à "couper" les branches de l'arbre de recherche. D'autre part les transitions qui aboutissent dans des classes supérieures ou égales b_{sup} ne sont pas introduites.

Dans l'exemple de la figure 9.11, le premier plan trouvé est le meilleur. Ce filtrage permet d'arrêter les recherches dès le niveau suivant car le graphe est déclaré saturé. Ainsi le graphe n'est développé que jusqu'au niveau 7 au lieu du niveau 9 qui est le niveau normal de saturation du graphe.

En effet, les différences existantes entre les niveaux 6 et 7 dans le graphe de planification de `Graphplan`, sont spécifiques à des plans dont les coûts sont supérieurs à b_{sup}. Etant mécaniquement mises à l'écart par le filtrage pour cette raison, il n'y a plus de différence entre les deux niveaux et le graphe apparaît saturé.

Les figures 9.12 et 9.13 présentent l'état initial et final d'un autre problème ou le meilleur plan n'est pas le premier trouvé.

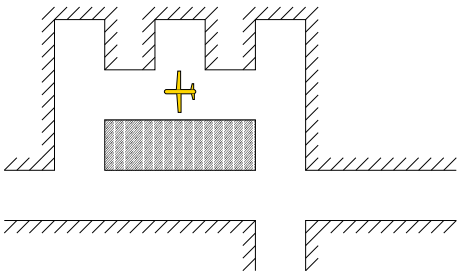


FIG. 9.12 – *Etat initial*

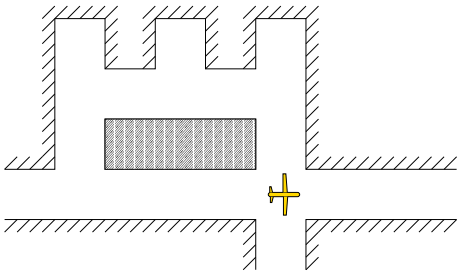


FIG. 9.13 – *Etat final souhaité, l'orientation de l'avion au final est sans importance dans ce problème. Seule sa position au tronçon* 6 *est requise.*

Le premier plan (cf. figure 9.14) est trouvé en développant le graphe sur 5 niveaux. Il a un coût de 9, à cause de l'utilisation d'un demi-tour. Le meilleur plan (cf. figure 9.15)est trouvé en 8 niveaux, avec un coût de 8 (aucun demi-tour n'a besoin d'être mis en œuvre). Aucun plan de coût intermédiaire n'est trouvé entre les deux.

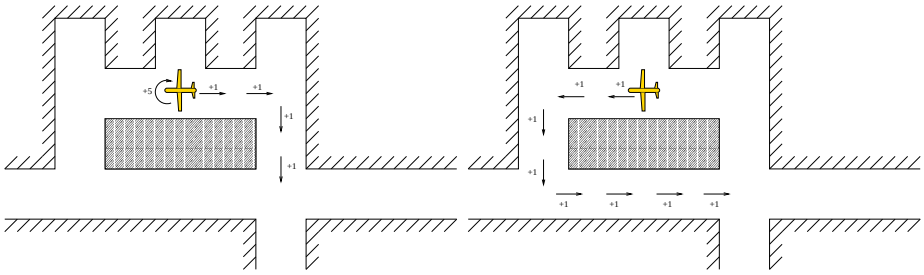


FIG. 9.14 – *Premier plan trouvé, en 5 niveaux, avec un coût de* 9.

FIG. 9.15 – *Le meilleur plan trouvé, en* 8 *niveaux, avec un coût de* 8

La figure 9.16 rassemble des données permettant de mieux appréhender l'impact du filtrage dans la phase de construction du graphe sur la taille de ce dernier. Elle présente le nombre de jetons qui sont concernés par le filtrage. Le total des "propositions par *NoOp*" correspond aux propositions qui doivent être recopiées par *NoOp*. Certaines ("réduites") ne seront pas recopiées dans toutes les classes prévues : l'ensemble des classes des jetons correspondants est réduit aux classes inférieures à b_{sup}. Parmi elles, certaines ("éliminées") ne seront pas recopiées du tout, le filtrage interdisant toutes leurs classes. Les mêmes données sont rassemblées pour les transitions. La "réduction" des ensembles de classes des jetons de transition concerne les classes de leurs effets.

Jusqu'au niveau 5, le filtrage n'a aucun effet puisqu'aucune borne supérieure n'est encore définie. A ce niveau, le premier plan est trouvé, permettant au filtrage d'avoir ses effets dès le niveau suivant. Le filtrage concerne presque tous les jetons, au niveau des places comme au niveau des transitions.

A chaque niveau, `Tokenplan` manipule un seul jeton par proposition ou par action considérant les ensembles de classes auxquels ils appartiennent. C'est pourquoi le filtrage élimine peu de jetons mais transforme presque tous ces ensembles. La figure 9.17 reprend les mêmes résultats en considérant cette fois les couples *(jeton, classe)*. Elle est plus représentative de la taille de l'espace de recherche liée à l'optimisation.

Il apparaît clairement que le filtrage arrête brusquement la croissance de la "largeur" de l'espace de recherche considéré. Après que le premier plan est trouvé, le graphe devient plus petit qu'il ne l'était avant et le reste jusqu'à sa saturation.

Du niveau 5 au niveau 6, le nombre d'actions applicables passe de 194 à 254. Cela traduit l'expansion de la surface de l'aéroport explorée par la recherche, malgré les restriction du filtrage. Cependant, dans le même temps, le nombre de transitions vers des classes différentes est en diminution. En fait, il y a plus de transitions déclenchables, mais à partir de moins de classes différentes - certaines d'entre elles ayant été éliminées par le filtrage.

C'est un cas où se compensent plus ou moins deux mécaniques. D'une part la "mécanique de la planification" détermine ce qu'il est possible ou non de faire. Elle est à ce moment-là toujours en découverte de nouvelles opportunités. D'autre part, la "mécanique de l'optimi-

	Avec filtrage						Sans filtrage	
	Propositions par *NoOp*			Transitions				
niveau	total	éliminées	réduites	total	éliminées	réduites	propositions	transitions
1	61	0	0	6	0	0	61	6
2	67	0	0	34	0	0	67	34
3	69	0	0	77	0	0	69	77
4	72	0	0	128	0	0	72	128
5	75	0	0	194	0	0	75	194
6	78	1	76	254	54	238	78	261
7	79	0	77	319	87	292	82	333
8	82	0	79	399	144	384	83	404
9	83	1	83	399	203	399	84	425

FIG. 9.16 – *Lorsque le filtrage est utilisé, les deux première colonnes de ce tableau indiquent le nombre de jetons (propositions ou transitions) concernés. C'est le nombre de jetons pour lesquels l'ensemble de classes associé est réduit, dont ceux pour lesquels l'ensemble de classes est réduit à l'ensemble vide (jetons éliminés). La troisième colonne donne pour référence le nombre de jetons générés sans filtrage.*

	Avec filtrage				Sans filtrage	
	Propositions par *NoOp*		Transitions			
niveau	total	éliminées	total	éliminées	propositions	transitions
1	61	0	6	0	61	6
2	249	0	40	0	249	40
3	572	0	189	0	572	189
4	926	0	823	0	926	823
5	1287	0	1727	0	1287	1727
6	1663	1031	1332	702	1663	2978
7	1016	371	1611	888	2056	4592
8	1039	385	1832	1068	2464	6607
9	1059	484	1496	987	2885	8837

FIG. 9.17 – *Les données présentées par ce tableau sont les mêmes que celles du tableau précédent. Cependant, ici, la factorisation des classes par jeton telle qu'elle est utilisée par* `TokenPlan` *est annulée. Ainsi, si un jeton est présent dans n classes différentes, il est comptabilisé ici n fois, alors qu'il ne l'était qu'une seule fois dans le tableau précédent. Ainsi, ce tableau donne la réalité du développement de l'espace de recherche alors que le précédent ne s'attachait en fait qu'aux labels différents.*

sation" désigne ce qu'il est intéressant ou non de faire. A ce moment-là, elle se contraint brusquement car elle vise à obtenir un meilleur résultat que le plan qui vient d'être trouvé.

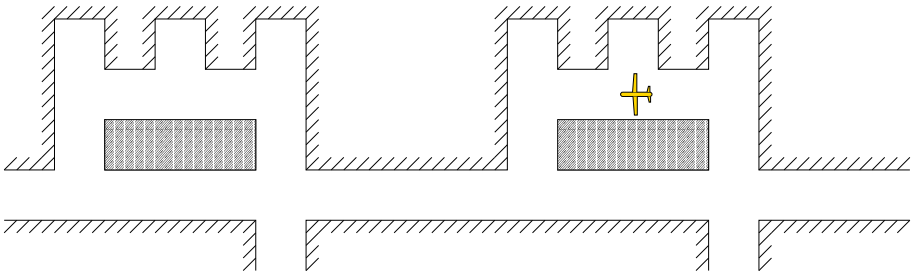


FIG. 9.18 – *Etat initial*

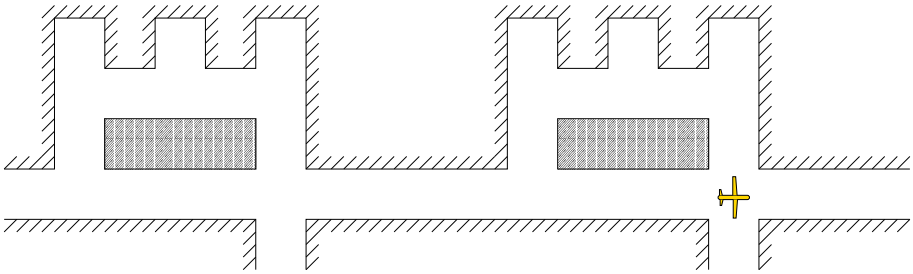


FIG. 9.19 – *Etat final souhaité, l'orientation de l'avion au final est dans importance dans ce problème. Seule sa position au tronçon* 6 *est requise.*

Enfin, notons que l'arrêt de l'expansion du graphe de planification est le fruit de cette cohabitation entre les logiques de planification et d'optimisation. Dans notre exemple, les objectifs d'optimisation créent une sorte de zone pertinente dont en sortir présenterait un coût trop important pour permettre de trouver le meilleur plan. En conséquence, la planification s'arrête quand le graphe de planification arrive à saturation **à l'intérieur de cette zone pertinente**.

Les figures 9.18 et 9.19 présentent les états initial et final d'un problème similaire à celui de notre exemple. La seule différence est que l'aéroport est deux fois plus grand. Dans ce cas, l'extension du graphe de planification s'arrête après le même nombre de niveaux, bien que cet arrêt ne permette pas d'envisager toutes les possibilités de déplacements offertes par cette extension de l'aéroport. En effet, l'essentiel de cette extension se trouve en dehors de la zone de pertinence du problème d'optimisation posé.

9.4 `TokenPlan` et programmation dynamique

La programmation dynamique est une approche souvent utilisée pour résoudre des problèmes d'optimisation. Nous montrons ici de quelle manière un même problème d'optimisation peut être traité par la programmation dynamique et par `TokenPlan`. Nous mettons aussi en évidence les situations dans lesquelles `TokenPlan` semble prometteur par rapport à la programmation dynamique.

9.4.1 Un domaine de planification commun pour l'expérimentation

Le problème que nous avons choisi pour notre expérimentation est un problème générique de robotique mobile. Bien que simple à comprendre, il prend en compte des contraintes importantes en situation réelle.

Un robot mobile doit se déplacer d'une position initiale I à une position but B en optimisant un critère spécifique. Nous définissons ce critère plus loin.

L'environnement est discrétisé en une grille de n lignes par m colonnes. Certaines cases sont inaccessibles car ce sont des obstacles. D'autres contiennent des amers de telle sorte que le robot peut s'y relocaliser (c'est-à-dire que le robot peut y déterminer sa position exacte).

Le robot dispose d'une primitive de déplacement lui permettant de passer d'une case à une case adjacente - au nord, au sud, à l'est, ou à l'ouest - si elles sont connectées l'une à l'autre dans le sens envisagé. Lors de chaque déplacement l'incertitude de positionnement du robot croît. Pour simplifier notre modèle, elle est incrémentée de 1 à chaque déplacement. Lorsque le robot se relocalise (à l'arrêt, dans une case contenant un amer), son incertitude est ramenée à 0.

Pour chaque problème, un seuil d'incertitude est défini à partir duquel le robot ne peut plus se déplacer. Concrètement, cela peut être assimilé à une sorte d'automatisme qui met le robot en mode "sécurité" en attendant une éventuelle intervention de récupération, quand la méconnaissance de sa position s'avère dangereuse pour lui-même ou pour l'environnement. Dans le cadre de la programmation dynamique, ce seuil permet aussi d'avoir un nombre d'états fini. C'est nécessaire pour générer la matrice de transitions utilisée par l'algorithme de résolution. Pour `TokenPlan`, ce seuil pourrait être infiniment grand puisque les états ne sont pas listés à l'avance.

Le critère à optimiser est un compromis entre la longueur du chemin à parcourir et sa sécurité. Le risque d'entreprendre un déplacement élémentaire est défini comme dépendant de deux paramètres :

- d'une part l'incertitude de positionnement du robot au départ ;
- d'autre part la difficulté intrinsèque du terrain en termes de positionnement des obstacles.

Ces deux paramètres sont intimement liés dans le calcul du risque d'un déplacement. Prenons l'exemple d'un robot cylindrique de 50 centimètres de diamètre. Il lui sera plus dangereux de tenter de passer dans un couloir d'1 mètre de largeur avec une incertitude de positionnement

d'une cinquantaine de centimètres seulement, que de traverser la largeur d'un terrain de football avec une incertitude de 10 mètres au départ .

Soit $C_{risque}(s,a,s')$ le coût en termes de risques d'exécuter l'action de déplacement a dans l'état s (aboutissant à un état s').

Soit $R_{env}(s,a,s')$ le risque intrinsèque à l'environnement à exécuter l'action de déplacement a entre les positions courantes dans les états s et s'. Dans notre exemple, il s'agissait de l'exiguité du couloir ou du caractère dégagé d'un terrain de football. Alors nous avons dans notre domaine de planification :

$$C_{risque}(s,a,s') = i(s) * R_{env}(s,a,s') \tag{9.1}$$

Nous définissons R_{env} (cf. 9.20) comme la somme du nombre de cases obstacles adjacentes aux cases de départ et d'arrivée. Celles adjacentes à l'arrivée sont pondérées par un facteur trois car il plus dangereux que l'arrivée soit étroite que le départ. Après tout, si le robot doit sortir d'un couloir il sait au moins qu'il est entre les murs au départ. S'il doit entrer dans un couloir, il se peut qu'il ne soit même pas en face!

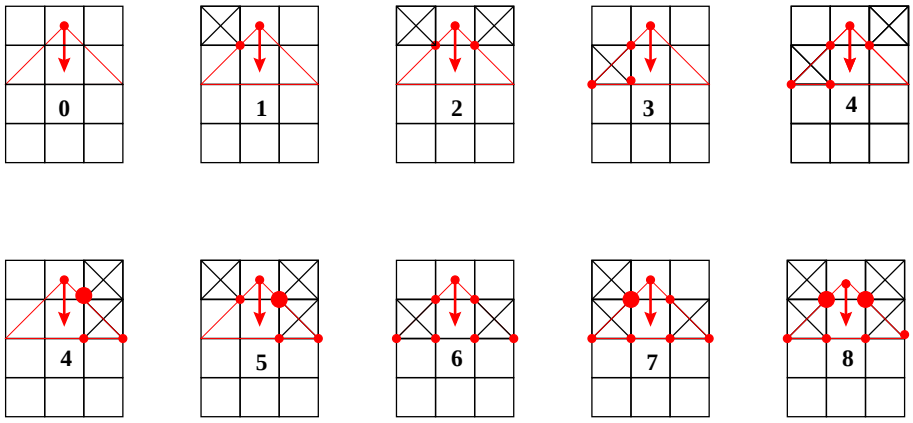


FIG. 9.20 – *Représentation de R_{env}. La flèche représente le déplacement envisagé. Les quatre cases qui entourent les cases de départ et d'arrivée sont considérée pour le calcul de R_{env}. Un obstacle à gauche ou à droite de la position de départ compte pour* 1. *Un obstacle à gauche ou à droite de la position d'arrivée compte pour* 3.

Au final, le coût à minimiser est la somme des coûts élémentaires définis comme la somme de la longueur d'un déplacement élémentaire et le risque de ce déplacement.

$$C(s,a,s') = 1 \ + \ C_{risque}(s,a,s') \tag{9.2}$$

L'étude des incertitudes et des risques pour eux-mêmes n'est pas l'objet de cette expérimentation. Bien-sûr, nos équations pourraient être remplacées par des modèles plus fins sans que cela change notre propos - à savoir que le morcellement et sa mise en œuvre au sein de `TokenPlan` permettent une grande expressivité des domaines de planification et le traitement de problèmes d'optimisation de ce type.

Pour générer aisément des environnements de test, nous avons développé une interface en Java. Elle permet de positionner les obstacles et amers en quelques clics. Le calcul des risques locaux (R_{env}), et la transcription de l'environnement sont réalisés automatiquement.

La figure 9.21 montre un exemple de problème de planification issu de ce domaine. Elle présente aussi une solution.

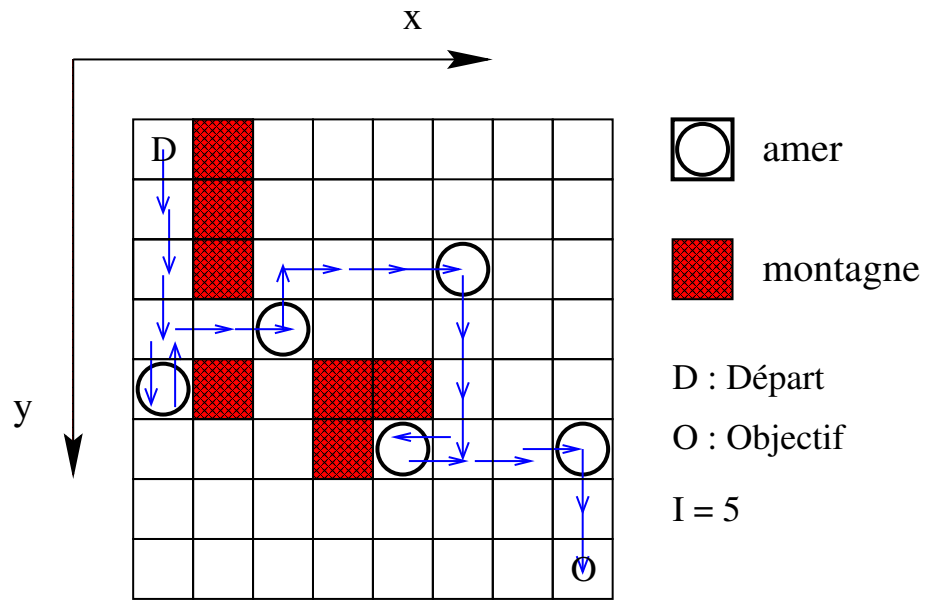


FIG. 9.21 – *Voici un exemple de problème - dans un environnement* 8x8 *- ainsi que sa solution. Certains allers et retours vers et depuis une zone avec un amer peuvent sembler étranges. En fait ils sont liés au seuil d'incertitude à ne pas dépasser, ici fixé à* 5*. Si ces détours pour se recaler n'étaient pas effectués, l'incertitude de positionnement du robot deviendrait si grande qu'il serait immobilisé. Il ne pourrait plus alors rejoindre l'objectif.*

9.4.2 Traitement par la programmation dynamique

Des problèmes de ce domaine de planification ont été résolus par un algorithme d'itération de la valeur (cf. section 2.2.1 de la première partie). Des détails sur cette implémentation peuvent être trouvés dans [70].

Cet algorithme est conçu pour résoudre ce type de problèmes. Le critère $C(s,a,s')$ que nous avons défini est adapté aux équations d'optimalité de Bellman. Soit V^{k+1} le vecteur d'états de l'itération $k+1$, $V^{k+1}(i)$ sa i-ème composante et S_a^i l'état atteint lorsque l'action a est exécutée dans l'état i. On a :

$$V^{k+1}(i) = \min(V^k(i),\ \min_a(V^k(S_a^i) + C(i,a,S_a^i))) \tag{9.3}$$

Avec cet algorithme, résoudre un problème de notre domaine de planification nécessite la construction préalable d'une matrice de transitions. Soient $nbCol$ le nombre de colonnes de la grille-environnement, $nbLig$ le nombre de lignes de la grille-environnement, et I le niveau d'incertitude maximal autorisé, alors le nombre d'états que cette matrice va devoir prendre

en compte est :

$$nbCol * nbLig * I$$

Par exemple, pour une grille $5X5$ avec 5 niveaux d'incertitudes ($I = 5$), cela représente 125 états - soit 125^2 ($= 15625$) transitions possibles. En conséquence, seuls les problèmes de petite taille peuvent être traités.

9.4.3 Traitement par `TokenPlan`

La transcription du domaine de planification pour `TokenPlan` est présentée dans la figure 9.23. La syntaxe est fondée sur celle de PDDL1, avec les adaptations que nous avons apportées pour

- traiter aisément les opérations numériques (PDDL2 n'était pas disponible au moment des expérimentations) ;
- spécifier un morcellement adapté à l'optimisation du critère défini (cf. section 7.7) ;
- tirer parti d'appels à des fonctions auxiliaires pour alléger le domaine de planification.

Le troisième point est d'importance dans ce type de domaines. Par exemple, la topologie (l'emplacement des amers et la connectivité des zones) est utilisée par la planification mais elle restera statique pendant tout le déroulement du plan. Le planificateur a besoin d'avoir accès à la topologie, pour information, et non pas comme objet de sa planification. Nous avons choisi de ne pas la représenter par des prédicats mais au travers d'appels à des fonctions. Celles-ci exploitent une représentation matricielle de la carte de l'environnement. La matrice `topo-tr-matrix` indique pour tout couple de positions si elles sont connectées et la contrainte de risque associée. Ainsi, nous évitons un nombre important de propositions à recopier d'un niveau à l'autre.

La figure 9.22, montre ce domaine de planification. Il peut être comparé à un domaine plus commun sans fonctions (figure 9.23) .

```
(define (domain duet-fct-typed)
      (:requirements :typing)
      (:types   (int numberp))  ;;int stands for intensional
                                ;; the fct numberp will have to be applied
                                ;; to check this type
      (:predicates (free ?x ?y - (int numberp))
                   (position ?x1 ?y1 - (int numberp)) ;; position of the robot
                   (uncertainty ?u - (int numberp)) ;; position uncertainty
                   (uncertainty_threshold ?t - (int numberp)))

      (:action move
          :parameters (?x1 ?y1 ?x2 ?y2 ?u ?t - (int numberp))
          :precondition (and (position ?x1 ?y1)
                             (free ?x2 ?y2)
                             (uncertainty ?u)
                             (uncertainty_threshold ?t)
                             (< ?u ?t)
                             (fct (connected-zones ?x1 ?y1 ?x2 ?y2)))

          :effect (and
                        (position ?x2 ?y2)
                        (uncertainty (+ ?u 1))
                        (free ?x1 ?y1)
                        (not (position ?x1 ?y1))
                        (not (free  ?x2 ?y2))
                        (not (uncertainty ?u)))
          :class (:split-fct (expr (my-criteria ?u ?x1 ?y1 ?x2 ?y2))))

      (:action relocate
          :parameters (?x ?y ?u - (int numberp))
          :precondition (and (position ?x ?y)
                             (uncertainty ?u)
                             (fct (is-amer ?x ?y)))
          :effect (and
                        (uncertainty (- ?u ?u))
                        (not (uncertainty ?u)))))

(defun MY-CRITERIA (class uncertainty x1 y1 x2 y2)
(let ((security-cstrt
        (aref  topo-tr-matrix (+ x1 (* y1 nb-lig)) (+ x2 (* y2 nb-lig)))))
   (if (null security-cstrt) ;; means positions (x1,y1) and (x2,y2) are not connected
        99999                ;; in this case, the cost should be infinitly high
        (+ class  5     (* uncertainty security-cstrt))
          )))
(defun CONNECTED-ZONES (x1 y1 x2 y2)
   (aref  topo-tr-matrix (+ x1 (* y1 nb-lig)) (+ x2 (* y2 nb-lig))))
(defun IS-AMER (x y)
   ( =  (aref carte x y ) 2))
```


FIG. 9.22 – *Domaine de planification utilisé par* **TokenPlan**. *Les fonctions LISP* **my-criteria**, **connected-zones** *et* **is-amer** *transcrivent le critère à optimiser, la connectivité des zones et l'emplacement des amers.*

```
(define (domain duet-typed)
      (:requirements :typing)
      (:types   zone
                (int numberp))  ;;int stands for intensional
                                ;; the fct numberp will have to be applied
                                ;; to check this type
      (:predicates (free ?z - zone)
                   (amer ?z - zone)
                   (connected ?z1 ?z2 - zone)  ;;from ?z1 to ?z2
                   (security_cstrt ?z1 ?z2 - zone ?sc - (int numberp))
                   (position ?z - zone) ;; position of the robot
                   (uncertainty ?u - (int numberp)) ;; position uncertainty
                   (uncertainty_threshold ?t - (int numberp)))
      (:action move
          :parameters (?z1 ?z2 - zone ?u ?t ?sc - (int numberp))
          :precondition (and (position ?z1)
                             (free ?z2)
                             (connected ?z1 ?z2)
                             (uncertainty ?u)
                             (uncertainty_threshold ?t)
                             (security_cstrt ?z1 ?z2 ?sc)
                             (< ?u ?t))
          :effect (and
                        (position ?z2)
                        (uncertainty (+ ?u 1))
                        (free ?z1)
                        (not (position ?z1))
                        (not (free ?z2))
                        (not (uncertainty ?u)))
          :class (:split-fct (expr (my-criteria ?u ?sc))))

      (:action relocate
          :parameters (?z - zone ?u - (int numberp))
          :precondition (and (position ?z)
                             (amer ?z)
                             (uncertainty ?u))
          :effect (and
                        (uncertainty (- ?u ?u))
                        (not (uncertainty ?u)))))
)
(defun MY-CRITERIA (class uncertainty security-cstrt)
   (+ class  1 (* uncertainty security-cstrt)))
```

FIG. 9.23 – *Domaine de planification utilisé par* ***TokenPlan****. La syntaxe est essentiellement celle de PDDL1, avec quelques adaptations que nous avons apportées pour traiter d'une part les calculs numériques (PDDL2 n'était pas disponible au moment des expérimentations), et d'autre part un morcellement adapté à l'optimisation du critère défini.*
La fonction LISP ***my-criteria*** *est la transcription du critère à optimiser. Elle doit être chargée dans l'environnement de* ***TokenPlan***

Arrêtons-nous un instant sur les prédicats :

- les obstacles sont des zones z pour lesquelles `(free z)` est faux dès l'état initial et qui ne sont pas la position initiale du robot ;
- le prédicat (*uncertainty_threshold* `?t - (int numberp)`) donne la valeur de I.

Les données nécessaires à l'optimisation sont encodées par le morcellement. En effet, les numéros des classes servent à accumuler les coûts élémentaires au fur et à mesure des déplacements du robot. Ainsi, à chaque fois que l'action `move` est introduite dans le graphe de planification, ses effets sont dans une classe supérieure à celle des préconditions. La différence entre les deux classes est calculée par la fonction `my-criteria` - qui correspond au critère à optimiser - suivant l'incertitude de positionnement utilisée comme hypothèse pour le déplacement, ainsi que la classe des préconditions. La classe des préconditions correspond au coût d'un plan (s'il existe) qui permet de les obtenir à partir de l'état initial.

Il est important de noter que le critère est monotone. L'incertitude de positionnement du robot peut subitement revenir à 0 lorsqu'il se relocalise, mais le coût global du plan ne peut jamais diminuer au fur et à mesure de son allongement. En conséquence, `TokenPlan` peut exploiter ses procédés de filtrage des classes pour appliquer une stratégie de type *Branch and Bound*.

Enfin, bien que `TokenPlan` permette de traiter des plans parallèles, les solutions seront ici toutes séquentielles. En effet, un seul robot est considéré. Celui-ci a seulement deux opérateurs à sa disposition, lesquels sont mutuellement exclusifs.

9.4.4 Un ensemble de cas à l'avantage de `TokenPlan`

Les approches de type programmation dynamique établissent pour toute position de départ possible le meilleur plan jusqu'à la position objective (ou l'inverse suivant le sens de cheminement de l'algorithme). Pour ce faire, elles exploitent la matrice des transitions possibles entre tous les états.

En conséquence, elles sont extrêmement sensibles à la taille de l'espace d'états. En revanche, pour un objectif donné, quelque soit le point de départ souhaité, l'effort fourni pour trouver la solution - et donc le temps nécessaire - seront les mêmes.

Le comportement de `TokenPlan` est tout autre. Il est insensible a priori à la taille de l'espace d'états puisqu'il l'explore de proche en proche. D'ailleurs, celui-ci peut être infini, ce qui n'est pas le cas pour les approches de programmation dynamique.

Du coup, si le plan solution est court par rapport à la taille de l'environnement, `TokenPlan` le trouve beaucoup plus rapidement qu'une approche de programmation dynamique (cf. figure 9.24). En effet, d'une part le développement des premiers niveaux du graphe de planification est léger quelle que soit la taille de l'environnement. D'autre part, dès qu'un plan a été trouvé et confirmé comme étant le meilleur, le planificateur s'arrête. Par ailleurs, dès qu'un plan a été trouvé, sa valeur permet un filtrage massif des classes et des actions à inclure dans l'extension du graphe de planification. Il s'ensuit que la progression de l'exploration de l'espace de recherche est allégée, jusqu'à la découverte du meilleur plan.

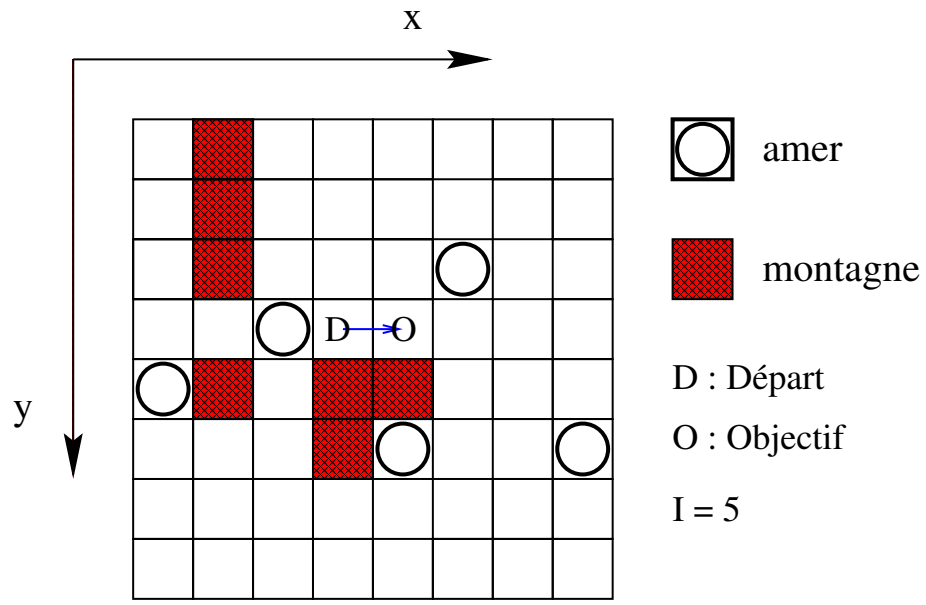


FIG. 9.24 – *Dans ce problème, l'objectif est une case adjacente de la case de départ. Le plan solution est trouvé extrêmement vite par* `TokenPlan` *puisqu'il apparaît dès le premier niveau d'actions du graphe de planification. En revanche, l'utilisation de l'itération de la valeur nécessite autant de temps que si l'objectif et le point de départ étaient extrêmement éloignés.*

La figure 9.25 compare les résultats obtenus en termes de temps CPU par `TokenPlan` et l'algorithme de programmation dynamique. Les performances de `TokenPlan` sont globalement moins bonnes que celles de la programmation dynamique. Cependant, elles montrent deux profils clairement différents :

- d'une part, pour un environnement donné, l'algorithme de programmation dynamique affiche des performances similaires quelque soit le plan solution du problème spécifiquement considéré ;
- d'autre part, `TokenPlan`, pour un environnement donné, trouve plus rapidement les plans solutions courts que les plans solutions longs.

Dimension de l'environnement	Longueur du plan	TokenPlan	Programmation Dynamique
5	1	0	0.03
5	7	22.51	0.05
5	8	41.41	0.04
5	8	44.96	0.04
7	1	0.01	0.18
7	5	3.59	0.2

FIG. 9.25 – *La première colonne indique la taille de l'environnement (le nombre de case par côté). Ainsi, les premières lignes concernent un environnement de* 5x5. *La deuxième colonne indique la longueur du plan solution. Les deux dernières colonnes indiquent le temps CPU utilisateur en secondes.*

9.4.5 Extensions de ce domaine d'expérimentation

Ce domaine est une première étape vers un domaine mettant en jeu la coopération de deux agents - c'est pourquoi sa transcription PDDL s'appelle *duet*. Un agent aérien serait ajouté à l'agent terrestre. La connaissance de la carte pourrait être incomplète. L'agent aérien aurait un champ de vision plus grand (ou une carte plus étendue mais moins précise).

La coopération des deux agents permettrait :
- à l'agent aérien d'informer à l'avance l'agent au sol sur les possibilités ou difficultés de déplacement à venir, que le robot terrestre ne peut pas encore percevoir depuis le sol ;
- à l'agent au sol de renseigner l'agent aérien des possibilités d'atterrissage
- aux deux agents de se relocaliser partiellement l'un sur l'autre (l'agent le plus incertain quant à sa position utilise l'autre comme sorte d'amer) ;
- éventuellement de permettre des franchissements particuliers (l'agent aérien emportant l'agent terrestre par dessus des obstacles, l'agent au sol portant l'agent aérien au travers de tunnels dans les obstacles trop hauts).

L'ajout d'un deuxième agent augmente la taille de l'espace d'états jusqu'à son carré. La taille de la matrice de transitions croît donc grandement, pénalisant encore plus les approches de programmation dynamique par rapport à la recherche "focalisée" de `TokenPlan`.

Par ailleurs, `TokenPlan` pourra prendre en compte les actions parallèles des deux agents. Ce n'est pas le cas des approches de programmation dynamique. La figure 9.26 montre le domaine de planification modifié pour prendre en compte plusieurs agents. Le critère global optimisé est alors une combinaison des valeurs individuelle du critère pour chaque robot. Il peut s'agir par exemple de la somme des valeurs individuelles, ou bien de la valeur individuelle maximale.

```
(define (domain duet2-fct-typed)
      (:requirements :typing)
      (:types   (int numberp)  ;;int stands for intensional
                               ;; the fct numberp will have to be applied
                               ;; to check this type
                robot)
      (:predicates (free ?x ?y - (int numberp))
                   (position ?r ?x1 ?y1 - (int numberp)) ;; position of the robot ?r
                   (uncertainty ?r ?u - (int numberp)) ;; position uncertainty
                   (uncertainty_threshold ?t - (int numberp)))

      (:action move
          :parameters (?r ?x1 ?y1 ?x2 ?y2 ?u ?t - (int numberp))
          :precondition (and (position ?r ?x1 ?y1)
                             (free ?x2 ?y2)
                             (uncertainty ?r ?u)
                             (uncertainty_threshold ?t)
                             (< ?u ?t)
                             (fct (connected-zones ?x1 ?y1 ?x2 ?y2)))

          :effect (and
                        (position ?r ?x2 ?y2)
                        (uncertainty ?r (+ ?u 1))
                        (free ?x1 ?y1)
                        (not (position ?r ?x1 ?y1))
                        (not (free  ?x2 ?y2))
                        (not (uncertainty ?r ?u)))
          :class (:split-fct (expr (my-criteria ?u ?x1 ?y1 ?x2 ?y2))))

      (:action relocate
          :parameters (?r ?x ?y ?u - (int numberp))
          :precondition (and (position ?r ?x ?y)
                             (uncertainty ?r ?u)
                             (fct (is-amer ?x ?y)))
          :effect (and
                        (uncertainty ?r (- ?u ?u))
                        (not (uncertainty ?r ?u)))))

(defun MY-CRITERIA (class uncertainty x1 y1 x2 y2)
(let ((security-cstrt
        (aref  topo-tr-matrix (+ x1 (* y1 nb-lig)) (+ x2 (* y2 nb-lig)))))
   (if (null security-cstrt) ;; means positions (x1,y1) and (x2,y2) are not connected
        99999                ;; in this case, the cost should be infinitly high
        (+ class  5     (* uncertainty security-cstrt))
           )))

(defun CONNECTED-ZONES (x1 y1 x2 y2)
   (aref  topo-tr-matrix (+ x1 (* y1 nb-lig)) (+ x2 (* y2 nb-lig))))
(defun IS-AMER (x y)
   ( =  (aref carte x y ) 2))
```


FIG. 9.26 – *Domaine de planification prenant en compte plusieurs robots. Une variable ?r de type robot est introduite pour distinguer les différents robots du problème.*

Chapitre 10

Conclusion de la partie 2

Considérer le morcellement de l'espace de recherche est un éclairage pertinent sur les différences entre diverses approches. Par exemple `Graphplan` ne considère comme critère d'homogénéité des ensembles d'états que l'exécutabilité des actions. Ce morcellement (dit *nul*) est particulièrement efficace pour explorer les enchaînement d'actions possibles. Cependant il n'est pas adapté à l'optimisation.

L'optimisation, nécessite de considérer des ensembles d'états homogènes du point de vue de leur utilité. Le plus simple est souvent d'utiliser un morcellement *total* - qui distingue chaque état.

Parallèlement, le morcellement de l'espace de recherche est aussi un angle de conception de nouvelles approches de planification. Nous avons par exemple conçu des stratégies de morcellement compatibles avec l'optimisation de plans tout en préservant l'intérêt de la planification disjonctive pour générer ces plans.

Nous avons réalisé un planificateur permettant d'expérimenter nombre de stratégies de morcellement : `TokenPlan`. Fondé sur une approche disjonctive et utilisant une représentation à base de réseaux de Petri, `TokenPlan` construit l'espace de recherche correspondant à la stratégie de morcellement qu'il reçoit en paramètre.

Sa participation à la compétition de planificateurs d'AIPS'2000 a montré qu'il pouvait se comporter comme un planificateur disjonctif du type de `Graphplan`. D'autres expérimentations que nous avons réalisées montrent qu'il peut aussi traiter des problèmes d'optimisation - d'ordinaire traités par des approches de programmation dynamique par exemple. En particulier, nous avons pu mettre en œuvre une technique de type *Branch and Bound* dans un contexte de planification disjonctive.

En cela, nos expérimentations valident la pertinence de la notion de morcellement de l'espace de recherche pour décrire les approches de planification existantes et en construire de nouvelles. Elles montrent l'importance de l'impact du morcellement mis en œuvre sur le comportement global du planificateur.

En revanche, elles n'ont pas permis de mettre en avant des gains de performance manifestes. D'une part, cela s'explique en partie par l'implémentation de `TokenPlan` qui n'est optimisée en aucune manière. D'autre part, cela indique aussi que le morcellement n'est pas une *solution* mais plutôt un *point de vue*. Trouver un morcellement efficace n'est en effet pas

une tâche simple.

Un aspect particulièrement intéressant du morcellement de l'espace de recherche est son expressivité. En particulier, il peut être présenté en termes d'arborescence de l'espace de recherche, ou en termes de découpages en ensembles d'états. Les deux approches sont souvent duales. Dans l'implémentation même de `TokenPlan`, deux recherches arrières ont été conçues. La première est plus conforme à une vision arborescente de l'espace de recherche. La seconde est moins contrainte, comme peut l'être une stratégie de morcellement en termes de rassemblements d'états.

Conclusion globale

La contribution de nos travaux est constituée tout à la fois des résultats que nous avons présentés et de la clef qu'ils constituent vers de nouvelles voies de recherche en planification.

Bilan

Les travaux concernant la planification d'actions ont été motivés par diverses priorités : rapidité de décision, complexité des systèmes considérés, optimisation, prise en compte d'incertitudes, etc. Ces multiples angles d'investigation ont mené à des approches qui semblent parfois très différentes.

Nous avons montré qu'une source fondamentale de cette apparente divergence est la façon dont l'information est organisée pour son traitement, c'est-à-dire la manière dont l'espace de recherche est structuré. La notion d'état et sa manipulation est au centre de cette structuration.

Une grande partie des travaux en planification d'actions a conduit à abstraire la notion d'état afin de pouvoir manipuler la seule information nécessaire pour déterminer l'exécutabilité d'une action et son adéquation aux propriétés recherchées. D'après nous, ce qui est important dans la mécanique d'enchaînement des actions n'est pas l'individualité des états, mais les propriétés qui font qu'un ensemble d'états permet ou non l'exécution d'une action.

Lorsqu'il s'agit d'optimisation ou de traitement d'incertitudes, l'exploitation de représentations éprouvées de la valeur et de l'erreur ont nécessité de pouvoir distinguer avec précision les états les uns des autres. Parfois, c'est même les cheminements au sein des états qu'il convient de distinguer - un même état atteint de deux façons différentes n'ayant pas la même valeur.

Nous pouvons ainsi décrire de manière unifiée nombre d'approches existantes en les distinguant par la seule manière dont elles regroupent les états pour les manipuler. C'est ce que nous appelons *le morcellement de l'espace de recherche*. Certaines doivent pouvoir manipuler aisément les plus grands ensembles d'états homogènes possibles - elles mettent en œuvre un *morcellement nul*. D'autres veulent pouvoir accéder directement aux plus petites entités distinguables (les états, voire des instances d'états) - elles mettent en œuvre un *morcellement total*.

Nous avons défini le morcellement de l'espace de recherche en termes de classes de propriétés. Cette formulation nous a permis de montrer formellement que la variation du morcellement aboutit effectivement à la description des différentes approches de planification considérées.

Afin de valider notre point de vue sur le plan expérimental, nous avons conçu et réalisé le planificateur `TokenPlan`. Il est capable d'exécuter les mêmes mécanismes de construction de l'espace de recherche et les mêmes mécanismes de recherche dans cet espace, pour des espaces de recherche très diversement morcelés. La stratégie de morcellement de l'espace de recherche est un paramètre de `Tokenplan`. Nos expérimentations montrent que la simple variation de ce paramètre - toutes choses égales par ailleurs - permet effectivement d'obtenir des comportements caractéristiques d'approches différentes.

Plus qu'un moyen de reformuler des approches existantes dans un cadre unifié, la notion de stratégie de morcellement de l'espace de recherche est un critère de spécification d'approches nouvelles en planification. Ce critère est intéressant car il peut être rattaché à des sémantiques simples telles que le facteur de branchement d'un espace structuré en arbre, ou plus généralement à la notion de regroupements d'états. Il nous semble ainsi plus facile de décrire des approches hybrides à partir de celles existantes.

En effet, chaque approche tire tout ou partie de son efficacité dans son domaine de l'adéquation entre les ensembles d'états qu'elle manipule et les ensembles qu'il lui suffit de pouvoir distinguer. C'est pour cette raison que les planificateurs classiques se focalisent sur les propriétés des états alors que les systèmes d'optimisation s'attachent aux zones d'utilité. Toute motivation hybride mène alors à un redécoupage hybride des ensembles d'états manipulés. C'est ainsi qu'une stratégie de morcellement adaptée nous a permis d'exploiter des techniques d'optimisation telles que le *Branch and Bound* dans un contexte de planification classique disjonctive.

Ces résultats ne constituent pas le seul intérêt de ces travaux. En effet, ces derniers ouvrent de nombreuses perspectives.

Perspectives

Tout d'abord, au niveau de la plateforme d'expérimentation `TokenPlan`, les principales contributions à apporter couvrent trois volets :

1. Intégrer la syntaxe PDDL2. En particulier, il serait intéressant d'étudier l'exploitation des indications d'optimisation que PDDL2 permet de transcrire pour définir une sorte de morcellement par défaut.
2. Raffiner le traitement des couples de mutuelle exclusion. Dans la version actuelle de `TokenPlan`, seules les relations de mutuelle exclusion transclasses sont prises en compte. Dans le cadre d'un morcellement arborescent, il existe aussi des relations de mutuelle exclusion propres à certaines classes (donc à certaines branches). Les considérer peut être utile avec certains domaines.
3. Explorer d'autres choix algorithmiques concernant l'intégration des classes. En particulier, cette version de `TokenPlan` fusionne en un seul tous les jetons d'une place qui portent le même label, l'associant à une liste de classes au lieu d'une seule. Il est ainsi possible de savoir instantanément quelles sont les classes dans lesquelles une proposition donnée semble accessible. Nous avons listé les autres possibilités. L'une de ces possibilités alternatives est de fusionner tous les jetons de même classe de sorte qu'il soit possible, pour une classe donnée, de connaître facilement toutes les propositions qu'elle contient et les transitions qui y sont possibles. Cette dernière approche est peut-être plus adaptée à certains types de morcellement.

Le dernier point concernant `TokenPlan` nous amène à considérer les perspectives liées à l'exploitation de nouvelles stratégies de morcellement. Bien sûr, la conception de ces stratégies et leur exploitation est en soi un champ d'investigation particulièrement large. Nous mentionnons ici deux directions spécifiques.

La première est l'exploitation du morcellement par abstraction jusqu'au niveau des classes. Nous l'avons évoqué sans avoir l'opportunité de l'expérimenter. La construction du graphe de planification sert de moyen pour élever le niveau d'abstraction du domaine à celui des classes. La recherche d'un plan se déroule alors au niveau des transitions entre classes. Par exemple, si le domaine traité modélise les déplacements d'un robot mobile en termes de mouvements unitaires (*Avancer*, *Reculer*, *Tourner*, ...), nous pensons qu'il serait possible par ce biais de remonter au niveau des déplacements entre salles - plus adapté à certains problèmes.

La seconde direction qui nous semble prometteuse dans ce thème concerne les relations entre morcellement et encodage des relations de mutuelle exclusion. D'une part, en couplant le traitement des relations binaires de mutuelle exclusion à des statégies de morcellement adaptées, il est peut-être possible de prendre en compte aussi des relations de mutuelle exclusion d'arité plus élevée. D'autre part, l'exploitation du morcellement pour encoder tout ou partie des relations binaires de mutuelle exclusion peut permettre de rendre les graphes de planification plus précis à moindre coût. Ce serait intéressant pour les planificateurs qui utilisent les graphes de planification comme estimateurs heuristiques. Des travaux sont d'ores et déjà en cours sur ce sujet ([77], [76]).

Enfin, l'étude des stratégies de morcellement pour elles-mêmes nous semble prometteur. Nous pouvons par exemple étudier la superposition de plusieurs systèmes de classes associés à des sémantiques différentes. Il s'agirait par exemple de plusieurs critères à optimiser, ou encore de la gestion (centralisée) de systèmes multi-agents. Techniquement, au final il n'y aurait toujours qu'un ensemble de classes. Cependant, en ce qui concerne d'une part la conception du domaine, et d'autre part la recherche d'un plan, cette superposition sémantique pourrait apporter à la fois des aides et des difficultés.

Elle pourrait apporter des aides parce que cette superposition pourrait être vue au contraire comme une décomposition d'un espace complexe qui permet justement d'écrire une stratégie de morcellement adaptée. Elle pourrait amener des difficultés car le traitement des classes pourrait devoir être différencié suivant le système de classes. En particulier, dans le cadre d'une optimisation, les bornes supérieures ou inférieures nécessaires à une approche *Branch and Bound* pourraient être différentes pour chaque critère.

Par ailleurs, nous pensons qu'il serait intéressant de pouvoir générer automatiquement ou semi-automatiquement des stratégies de morcellement à partir du domaine à traiter et des objectifs du problème (optimisation, gestion de l'incertain, rapidité, ...). Les travaux de Long et Fox ([46], [45]) concernant la reconnaissance de structures particulières dans les domaines pourraient être utiles dans cette voie.

Dans ce but, des travaux sur la caractérisation des morcellements nous semblent nécessaires. En effet, suivant les morcellements, les classes sont plus ou moins nombreuses, se recoupent plus ou moins, etc. Il en découle des différences quant à la répartition des relations de mutuelle exclusion en relations transclasses et intraclasses, ainsi que des différences quant à la mémoire occupée, quant aux possibilités de filtrage, etc. Peut-être y aurait-il un moyen de caractériser une stratégie de morcellement par rapport au but qu'elle poursuit pour aider à

sa conception. Ce serait une démarche similaire à celle qui aide à la conception des relations d'une base de données relationnelle en déterminant si le *design* correspond à l'une des *formes normales* (cf. [17], section 12.3, pour une introduction aux *formes normales* des bases de données relationnelles).

Les travaux présentés dans ce manuscrit ouvrent de nombreuses voies. Elles sont d'autant plus prometteuses que le domaine de la planification arrive à une phase propice à la mise en commun des atouts des diverses approches. La notion de morcellement est un outil facilitant les combinaisons nécessaires dans cette optique.

Troisième partie

Annexes

Annexe A

Méthodes de recherche

Les principales méthodes de recherche sont présentées ici. Elles sont génériques car elles sont indépendante du domaine d'application. Les algorithmes dédiés à telle ou telle application (dont la planification d'actions) se fondent sur ces méthodes et leurs variations.

A.1 Notions de base

Une méthode de recherche est appliquée à un espace de recherche. A partir d'une zone déterminée de cet espace, il s'agit de le parcourir jusqu'à atteindre ce qui est recherché.

Par exemple, le problème peut être de trouver une image montrant des étudiants. L'espace de recherche peut-être l'Internet, et la recherche initiée sur la page d'un moteur de recherche. La recherche consiste alors à suivre les liens hypertexte jusqu'à trouver une page avec une photo adaptée. L'espace de recherche peut aussi être une ville, dans laquelle nous nous déplaçons en voiture pour mener la recherche. Le quartier des universités nous permettra sans doute de prendre une photo d'étudiants.

Au niveau de la recherche, l'espace de recherche peut être représenté par un graphe. Chaque nœud correspond à un élément de l'espace (une page web ou un lieu, dans les exemples précédents). Les liens définissent l'accessibilité des éléments entre eux (ils correspondent aux liens hypertexte des pages web, ou aux rues de la ville).

Pour chaque nœud exploré, il est nécessaire que la méthode de recherche ait à sa disposition la liste des "voisins" de ce nœuds.

Définition : Successeurs

Les successeurs d'un nœuds N sont tous les nœuds qui peuvent être atteints en un seul coup à partir de N. Dans le graphe, ils sont reliés à N par un lien direct, partant de N.

Définition : Prédécesseurs

Les prédécesseurs d'un nœuds N sont tous les nœuds à partir desquels N peut être atteint en un seul coup. Dans le graphe, ils sont reliés à N par un lien direct, aboutissant à N.

Il n'est pas toujours facile de calculer le prédécesseur. En effet, les lois d'accessibilités ne sont pas toujours commutatives.

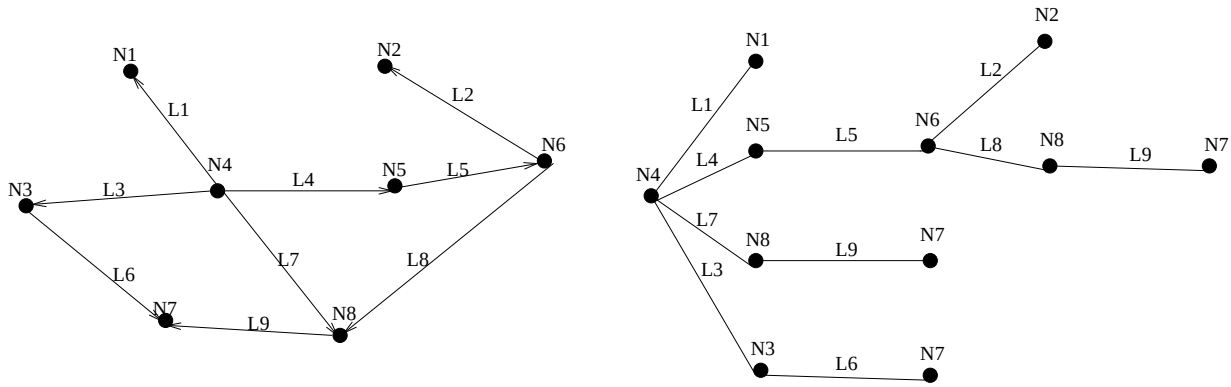


FIG. A.1 – *Recenser les chemins possibles à partir d'une origine dans un graphe revient à construire un arbre. Ici, nous avons choisi N4 comme origine.*

En parcourant l'espace de recherche, les méthodes de recherche "déroulent" des chemins possibles à partir de l'origine. Ce faisant, la plupart d'entre elles créent un arbre de recherche à partir de graphe de l'espace de recherche, comme le montre la figure A.1.

Définition A.1 : Facteur de Branchement

Le nombre de successeurs d'un nœud s'appelle son facteur de branchement. Le facteur de branchement est le plus souvent considéré de manière globale sur un arbre : en moyenne, ou au pire cas.

Les différences entre les stratégies de recherche présentées ci-après proviennent de leurs manières de choisir les pistes à explorer. Quand le nœud courant a plusieurs successeurs, faut-il en suivre un et abandonner les autres? Faut-il tous les essayer? Lequel devrait être exploré en premier? etc.

Quatre critères principaux permettent de comparer les différentes approches.

Critères de comparaison des méthodes de recherche

- *Complétude* : la méthode est dite complète si elle garantit de trouver ce qui est cherché lorsque l'espace de recherche le contient bien ;
- *Optimalité* : si ce qui est cherché peut être évalué, la méthode de recherche trouvera-t-elle la meilleure instance présente dans l'espace de recherche, ou seulement l'une quelconque des instances présentes?
- *temps de recherche* : Le temps ncessaire pour trouver une solution (si la dure de recherche n'est pas connue, il est pertinent de comparer le nombre de noœuds "explorés" par la recherche avant de trouver une solution) ;
- *mémoire nécessaire* : l'espace mémoire nécessaire à la recherche pour aboutir.

A.2 Recherches systématiques non informées

Il est question de recherche non informée lorsqu'aucune information n'est disponible quant à la position - même approximative - de la solution. L'espace de recherche est parcouru de manière systématique jusqu'à ce qu'une solution soit trouvée. Si une solution optimale est

recherchée, alors tout l'espace est parcouru, pour garantir que toutes les solutions soient trouvées. Il suffit alors de n'en garder que la meilleure.

A.2.1 Recherche en largeur d'abord

Une recherche *en largeur d'abord* ou *Breadth First Search* parcourt l'arbre de recherche niveau par niveau : tous les nœuds de profondeur 1, puis tous les nœuds de profondeur 2, et ainsi de suite (cf. figure A.2.1).

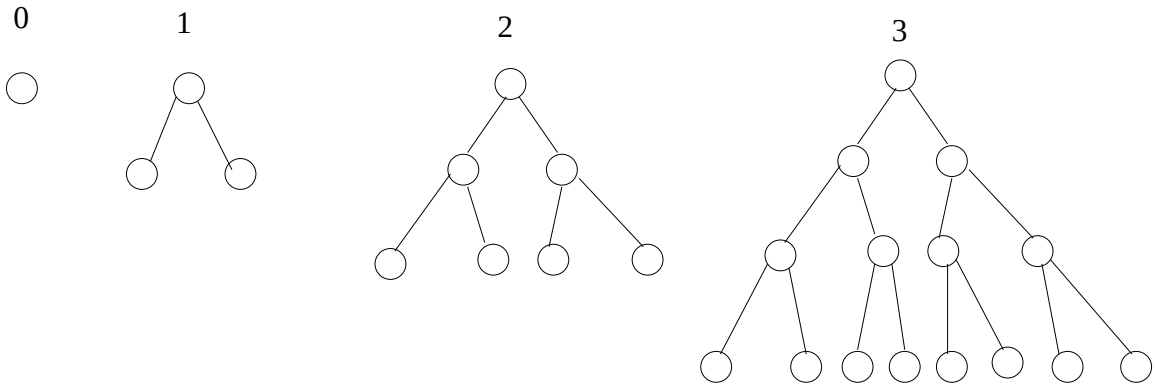


FIG. A.2 – *Premières étapes du développement d'un arbre de recherche en largeur d'abord.*

Valeur des quatres critères pour la recherche en largeur d'abord

- *Complétude* : complète ;
- *Optimalité* : optimale (l'optimalité de la solution n'est garantie qu'une fois que toutes les solutions ont été trouvées)
- *temps de recherche* : soit b le facteur de branchement et p la profondeur à laquelle apparaît la solution, $b^p + b^{p-1}$ nœuds sont développés;
- *mémoire nécessaire* : tout ce qui a été développé de l'arbre doit être gardé en mémoire - soient $b^p + b^{p-1}$ nœuds. C'est le défaut essentiel de cette approche.

Si le nœud solution est connu et que la recherche porte sur la manière de l'atteindre depuis le nœud origine, il est possible d'appliquer une recherche en largeur d'abord en arrière (*Backward Breadth First Search*). L'algorithme A.2.1 est alors appelé avec le nœud solution. Les *successeurs* sont remplacés par les *predecesseurs*.

Il est aussi possible de mettre en œuvre une recherche bidirectionnelle. Deux recherches sont lancées, l'une depuis le nœud origine, l'autre depuis le nœud solution. Le processus s'arrête dès qu'elles se rencontrent. Cette approche bénéficie à la fois de la focalisation sur l'origine de la recherche en avant et de la focalisation sur l'objectif de la recherche en arrière, ce qui limite le nombre de nœuds développés à $2b^{p/2}$ au lieu de b^p. La figure A.2.1 montre intruitivement d'où provient ce gain.

A.2.2 Recherche en profondeur d'abord

Une recherche *en profondeur d'abord* ou *Depth First Search* parcourt l'arbre de recherche branche par branche. A chaque branchement la première branche est suivie et ainsi de suite.

Algorithme 4: Algorithme de recherche arborescente en largeur d'abord (*Breadth First Search*). Le premier appel se fait avec le nœud origine.

```
BFS(listeNœuds)
si listeNœuds = ∅ alors
    retourner Echec
sinon
    prochainsNœuds ← ∅
    pour chaque nœud dans listeNœuds faire
        si solution(nœud) alors
            retourner nœud
        sinon
            prochainsNœuds ← successeurs(nœud)
    retourner BFS(prochainsNœuds)
```

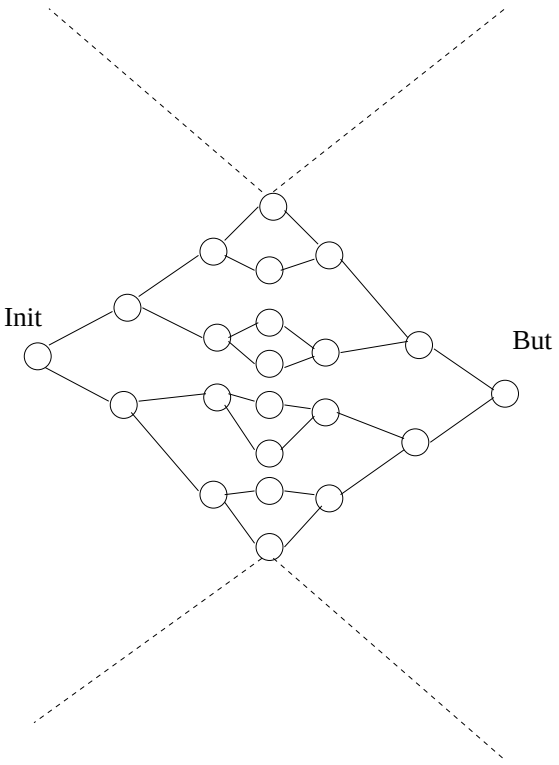


FIG. A.3 – *Visualisation schématique d'une recherche en largeur d'abord bidirectionnelle. Les errements (en pointillés) des recherches monodirectionnelles lorsqu'elles s'éloignent de leur point de départ sont remplacés par le début de l'autre recherche - encore focalisée.*

Quand un nœud n'a pas de successeur, alors le dernier branchement rencontré est considéré, et la première branche parmi celles non encore explorées est choisie. La figure A.2.2 montre les premières étapes du développement d'un arbre de recherche en profondeur d'abord.

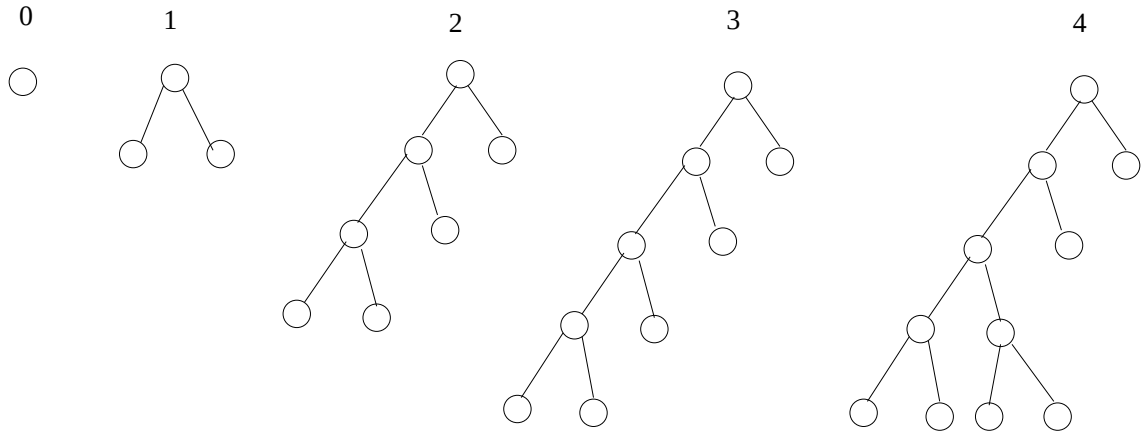


FIG. A.4 – *Premières étapes du développement d'un arbre de recherche en profondeur d'abord. Les nœuds développés à l'étape* 3 *n'ont pas de successeurs, alors à l'étape* 4 *le dernier nœud laissé en suspens est développé.*

Valeur des quatres critères pour la recherche en profondeur d'abord

- *Complétude* : incomplète. A cause de circuits dans le graphe de l'espace de recherche, certaines branches de l'arbre de recherche peuvent être infinies. Si une telle branche est explorée avant la première branche contenant un nœud solution, celui-ci ne sera jamais exploré. C'est le défaut principal de cette approche.
- *Optimalité* : non optimale (puisqu'incomplète)
- *temps de recherche* : soit b le facteur de branchement et m la profondeur maximale de l'arbre, de l'ordre de b^m nœuds sont développés;
- *mémoire nécessaire* : seule la branche en cours d'exploration est en mémoire, soit au pire bm nœuds.

Pour rendre cette recherche complète une profondeur maximale peut être spécifiée. L'arbre n'est pas développé au delà de cette limite. La difficulté est alors de régler cette profondeur limite. La choisir petite peut empêcher la solution d'être trouvée. La choisir trop grande peut laisser se perdre inutilement la recherche dans de très grandes branches alors que la solution est proche de la racine.

Ces ecueils sont contournés par la *recherche en profondeur itérée* (*Iterative Deepening Search*). Il s'agit simplement d'essayer toutes les profondeurs limites : 1, puis 2, puis 3 et ainsi de suite jusqu'à ce que la solution soit trouvée. Cette approche est complète et optimale - comme la recherche en largeur d'abord - mais se contente de faibles ressources mémoire - comme la recherche en profondeur d'abord. C'est la recherche la plus adaptée aux grands espaces de recherche.

Bien-sûr, les premiers niveaux sont à chaque fois redéveloppés. Cependant, il faut garder à l'esprit que le nombre de nœuds de chaque niveau est de l'ordre de $b - 1$ fois supérieur

Algorithme 5: Algorithme de recherche arborescente en profondeur d'abord (*Depth First Search*). Ici & est l'opérateur de concaténation de deux listes. Le premier appel se fait avec le nœud origine.

```
DFS(listeNœuds)
si listeNœuds = ∅ alors
    retourner Echec
sinon
    prochainsNœuds ← listeNœuds
    nœudCourant ← premier(listeNœuds)
    si solution(nœudCourant) alors
        retourner nœudCourant
    sinon
        prochainsNœuds ← successeurs(nœudCourant)&prochainsNœuds
    retourner DFS(prochainsNœuds)
```

au nombre de nœuds de tous les niveaux précédents (b étant le facteur de branchement). En conséquence, en regard du nombre de nœuds du dernier niveau développé, le développement itératif des niveaux précédents n'a que peu d'importance. Plus le facteur de branchement est grand, plus ce surcoût paraît petit.

Au pire - dans le cas d'un arbre binaire - le nombre de nœuds développés est environ le double de celui développé par une recherche en largeur d'abord. En revanche seuls bd nœuds ont besoin d'être mémorisés (b étant toujours le facteur de branchement, et d la profondeur de la solution).

A.3 Recherches systématiques informées

Il est question de recherche informée lorsque le parcours de l'espace de recherche est guidé par des informations sur la proximité d'une solution. Ainsi, toutes les branches de l'arbre de recherche n'ont plus a priori le même potentiel de succès. Les plus prometteuses sont explorées prioritairement. Il peut aussi être possible de faire l'impasse sur certaines branches, car il est certain qu'elles ne mèneront pas vers une solution.

Bien-sûr, ces jugements a priori sont fait à partir des informations disponibles. L'avantage (ou le désavantage) des recherches informées par rapport aux recherches systématiques non informées dépend donc de la qualité et de la pertinence de ces informations et de leur interprétation. C'est le rôle joué par l'*heuristique*.

A.3.1 Heuristique

Une *heuristique* est une méthode qui guide vers une solution sans garantir qu'en suivant ses indications une solution sera trouvée. Un exemple d'heuristique est la stratégie que met en un œuvre tout un chacun lorsqu'il cherche un cadeau de Noël. Suivant le destinataire du cadeau, il va d'abord commencer par un magasin de produits culturels ou une boutique de vêtements, par exemple, se disant qu'il a plus de chance de trouver une bonne idée-cadeau là que dans son magasin d'alimentation habituel. Cependant, ce dernier vend des boîtes de chocolats. Il

n'est donc pas impossible d'y trouver quelque chose d'intéressant. Ainsi, cette stratégie ne garantit pas de trouver immédiatement un cadeau, mais elle semble plus raisonnable que de visiter de manière systématique tous les commerces de la ville!

Dans le même ordre d'idée, les méthodes de recherche informées utilisent une *fonction heuristique*.

Définition A.3.1 : Fonction heuristique

Une *fonction heuristique* associe à chaque nœud N de l'espace de recherche une valeur telle que cette valeur soit nulle si N est un nœud solution. Elle est notée h.

Une fonction heuristique est définie de telle manière qu'elle guide la recherche dans l'espace de recherche vers les solutions. Lorsqu'il s'agit de trouver une solution quelconque elle indique la proximité des solutions par rapport au nœud courant.

Dans le cadre de problèmes d'optimisation. L'heuristique reprend généralement le critère à optimiser.

Par exemple, lorsque le problème est de trouver un itinéraire le plus court dans une ville entre un point A et un point B, la distance en ligne droite entre une position P et B peut être utilisée comme heuristique. Si $P = B$ nous avons bien $h(P) = 0$, et inversement. Dans le cas contraire, se diriger vers le voisin qui a la plus petite valeur heuristique contribue bien à se rapprocher de B.

Définition A.3.1 : *Heuristique admissible*

Une fonction heuristique est dite *admissible*, si elle ne surévalue jamais la distance au but.

Une heuristique admissible est optimiste. Elle n'indiquera jamais qu'une solution est plus lointaine qu'elle ne l'est réellement. L'heuristique de notre exemple est admissible. En effet, la distance la plus courte entre deux points est la ligne droite. En conséquence, aucun itinéraire ne peut être plus court. En revanche, il peuvent être beaucoup plus long. Il est même parfois avisé de ne pas suivre les conseils de l'heuristique (cf. figure A.5).

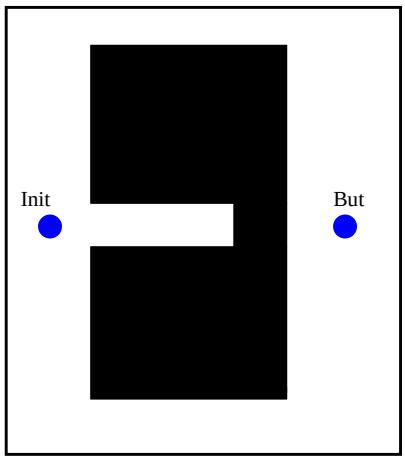


FIG. A.5 – *Exemple où notre heuristique, admissible, est de "mauvais conseil". L'heuristique est la distance du but en ligne droite. Dans cet exemple, l'heuristique suggère d'aller vers la droite. Cela revient à s'engouffrer dans l'impasse.*

A.3.2 Meilleur d'abord (*Best First Search*)

Une recherche "meilleur d'abord" choisit à chaque instant de développer parmi les nœuds atteints celui qui a la meilleure valeur heuristique.

Définition A.3.2 : ***Recherche meilleur d'abord***

Une recherche "meilleur d'abord" choisit à chaque instant de développer parmi les nœuds atteints celui qui a la meilleure valeur heuristique.

Une recherche "meilleur d'abord" se comporte comme une recherche en profondeur d'abord excepté pour le choix de la première branche à explorer.Elle en présente aussi les désavantages, comme l'incomplétude en particulier. Par ailleurs, même quand la recherche aboutit à une solution, celle-ci n'est pas nécessairement la meilleure: la recherche "meilleur d'abord" n'est pas optimale.

A.3.3 A^*

Définition A.3.3 : A^*

Une recherche A^* choisit à chaque instant de développer parmi les nœuds atteints le nœud n qui a la meilleure valeur $f(n) = g(n) + h(n)$.
$h(n)$ est la valeur heuristique de n - soit le coût estimé pour rejoindre le but à partir de n.
$g(n)$ est le coût exact du chemin entre la racine de l'arbre de recherche et n.

Contrairement à une recherche "meilleur d'abord", la recherche A^* est complète et optimale sous certaines conditions.

Condition de complétude de A^*

Une recherche A^* est complète si et seulement si les deux conditions suivantes sont remplies :

1. chaque nœud de l'arbre de recherche a un facteur de branchement fini ;
2. toute expansion d'un nœud a un coût minimal δ strictement positif.

Intuitivement, la seconde condition de complétude évite de suivre une branche infiniment grande comme les recherches en profondeur d'abord (cf. figure A.6). En effet, $g(n)$ ne pouvant que croître le long de cette branche (du fait de coût minimal non nul δ), la valeur $f(n)$ va nécessairement excéder une valeur $f(n')$ d'un nœud en attente de développement. Ce nœud n' devient alors le "meilleur" nœud et il est choisi pour être développé.

Condition d'optimalité de A^*

Une recherche A^* est optimale si et seulement si l'heuritique h utilisée est admissible (c'est-à-dire qu'elle ne surestime jamais le coût pour rejoindre le but).

Démonstration : A^* est optimale

Soit B un nœud but optimal - de coût f^*.
Soit B_2 un autre nœud but, suboptimal - $f(B_2) > f^*$.
Supposons que la recherche A^* retourne B_2 comme nœud but.
Soit N une feuille de l'arbre qui se situe sur le chemin optimal entre la racine et B. N existe nécessairement, à moins que l'arbre ait été développé jusqu'à B, mais dans ce cas B aurait été retourné au lieu de B_2.
$f(N) >= f(B_2)$ puisque B_2 a été sélectionné comme meilleure feuille.

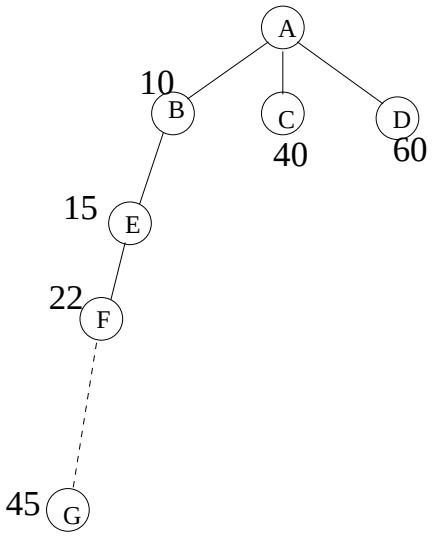


FIG. A.6 – *Même si une branche est infiniement longue, il arrive toujours un moment où elle devient moins intéressante que l'un des nœuds laissés de côté.*

$f^* >= f(N)$ puisque h est admissible.
Donc $f^* >= f(B_2)$.
Or $h(B_2) = 0$ puisque B_2 est un but.
En conséquence, $f^* >= g(B_2)$.
Ceci est en contradiction avec le fait que B_2 soit suboptimal.

A.3.4 *Branch and bound*

Branch and bound est un schéma générique de recherche de la meilleure solution développée dans le domaine de la recherche opérationnelle. Contrairement aux autres recherches informées présentées ici, il n'utilise pas forcément d'heuristique. En revanche, il nécessite de connaître a priori une borne supérieure du coût de la meilleure solution. Le coût d'une solution quelconque peut être cette borne. Par ailleurs, il est nécessaire de pouvoir déterminer une borne inférieure au coût de la meilleure solution d'un ensemble de solutions.

Le principe est le suivant. Soit S l'ensemble des solutions présentes dans l'espace de recherche. Soit b^S_{sup} une borne supérieure du coût de la meilleure solution. Considérons S_1 et S_2 deux sous-ensembles de S tels que $S_1 \cup S_2 = S$. Soit $b^{S_1}_{inf}$ (respectivement $b^{S_2}_{inf}$) une borne inférieure du coût de la meilleure solution présente dans S_1 (respectivement (S_2).

Si $b^{S_1}_{inf} > b^S_{sup}$, alors la meilleure solution de S_1 a un coût nécessairement supérieure à celui de la meilleure solution de S. Cela signifie que la meilleure solution de S n'est pas dans S_1. En conséquence, il n'est pas nécessaire de continuer les recherches dans S_1.

Si $b^{S_1}_{inf} < b^S_{sup}$ rien n'assure que la meilleure solution de S soit dans S_1, mais cela reste possible. S'il est possible de connaître une borne supérieure du coût de la meilleure solution de S_1 - soit $b^{S_1}_{sup}$, alors si $b^{S_1}_{sup} < b^S_{sup}$, b^S_{sup} prend pour nouvelle valeur $b^{S_1}_{sup}$. Le même raisonnement est mené pour $S2$.

Pour le ou les sous-ensembles qui restent pertinents, le même processus de découpage (*Branch*) et d'évaluation (*Bound*) est ré-itéré, et ainsi de suite. Le processus s'arrête quand

$b^S_{inf} = b^S_{sup}$.

Bien-entendu, la précision avec laquelle est borné le coût de la meilleure solution d'un ensemble joue un rôle primordial. Plus cette précision est grande plus il sera facile d'éliminer rapidement de grandes parties de l'espace de recherche.

Il ne faut pas non plus que l'évaluation des bornes nécessite trop de calculs. Lorsque le coût croît de manière monotone avec la distance à la racine de l'arbre, le coût de chaque nœud est une borne inférieure du coût de ses descendants. Les bornes supérieures peuvent n'être mises à jour qu'à chaque fois qu'une solution est trouvée.

Contrairement à une recherche A^*, une recherche *Branch and Bound* ne revient jamais sur les branches de l'arbre de recherche qu'elle a arrêté de développer. C'est dû au fait qu'elle n'utilise pas d'estimation heuristique mais des valeurs exactes.

A.4 Recherches locales

Définition : *Recherche locale*

Une *recherche locale* appréhende l'espace de recherche localement. Elle ne mémorise pas le parcours qu'elle a déjà effectué. Elle peut être amenée à visiter intégralement l'espace de recherche, mais à chaque instant elle ne prend en considération que les informations à sa disposition localement.

Nous n'abordons ici que la *Hill Climbing Search*. C'est une recherche locale basique, à partir de laquelle plusieurs variations ont été développées.

Ce type de recherche est utilisé pour les problèmes d'optimisation. En effet, chaque élément de l'espace de recherche doit être associé à une valeur. A chaque étape, les plus proches voisins de l'élément courant sont considérés. Celui qui présente la meilleure valeur sera le prochain élément courant.

Intuitivement, l'espace de recherche peut se représenter en trois dimensions. Chaque élément correspond à un point du plan (x,y). Sa valeur correspond à une hauteur. Dans le cadre de la minimisation de cette valeur (minimisation d'un coût par exemple), cette recherche se contente de suivre les plus fortes pentes.

S'il existe des minima locaux, la recherche peut y rester coincée, croyant que c'est le minimum de l'espace. Une technique simple consiste à relancer la recherche d'un autre endroit de l'espace choisi aléatoirement. Au final, la meilleure des solutions trouvées est retournée.

Lorsque la recherche aborde une zone dans laquelle tous les éléments ont la même valeur, le successeur est choisi aléatoirement. La recherche peut "errer" longtemps sur de tels "plateaux".

Cependant, l'avantage essentiel de cette recherche est de ne nécessiter qu'extêmement peu de mémoire. En effet, il n'y a pas d'arbre de recherche à maintenir, même partiellement. Seule la valeur de l'élément courant est prise en compte à chaque instant.

Annexe B

Planification de mouvements

La planification de mouvements est un domaine spécialisé. Il ne concerne pas directement les travaux présentés dans ce document. Toutefois, dans le développement de notre contribution, nous ferons un parallèle avec celui-ci afin d'aider à la compréhension de notre démarche. C'est dans ce seul but qu'une brève présentation est proposée ici. Ainsi, le lecteur préférent ne pas lire ce chapitre ne sera pas bloqué dans la suite du document.

La capacité de pouvoir planifier ses mouvements est de prime importance pour un système robotique autonome. De nombreux travaux ont donc été menés sur ce sujet (et le sont toujours), sous des angles très variés. La diversité des spécialités des équipes et des motivations de leurs études ont abouti à un grand nombre d'approches et d'algorithmes.

La présentation synthétique qui suit ne retient de ces approches que les plus significatives en matière d'adaptation de leur espace de recherche aux types de problèmes à résoudre. Elle est tirée d'une étude bibliographique plus complète présentée dans [52].

Après avoir présenté les hypothèses qui fondent la planification de mouvements, nous nous intéresseront aux méthodes de planification en boucle ouverte, qui forment en fait le noyau des méthodes classiques du domaine. Ensuite, nous nous focaliserons sur des approches en boucle fermée.

B.1 Tâche, agent et environnement

Classiquement, la planification de mouvements ne traite que de problèmes d'atteinte de but terminal. L'état initial est la configuration spatiale des objets physiques de l'environnement dans laquelle se trouve l'agent avant d'exécuter la tâche considérée, et l'état-objectif est la configuration à atteindre ([43, 34, 68]). De ce point de vue elle reste tout à fait semblable à la planification classique d'actions.

En ce qui concerne l'agent lui-même, les seules caractéristiques intrinsèques considérées sont cinétiques (degrés de liberté, vitesse maximale, holonomie). Ses capacités d'action sur l'environnement se limitent en général à son interaction avec les obstacles (Peut-il les déplacer? Peut-il les déformer ou les détruire?). Enfin, ses éventuelles aptitudes perceptives doivent lui permettre de recaler sa position, ou de découvrir l'environnement physique (en construire une carte).

B.2 Espace des configurations

La *configuration* d'un agent est représentée par un ensemble de paramètres indépendants qui déterminent totalement la position de tous les points de l'agent dans l'espace physique. Par exemple, dans le cas d'un agent en forme de triangle dans un espace physique en trois dimensions, une configuration pourra être décrite par six paramètres (3 pour la position d'un sommet quelconque du triangle, et 3 angles pour l'orientation de ce triangle). L'ensemble de ces configurations constitue *l'espace des configurations*.

Grâce à l'utilisation de cet espace (introduit dans [48]), tous les problèmes de planification de mouvements peuvent être ramenés à la recherche d'une suite de **points** connectés entre la configuration initiale et la configuration finale. Ainsi, la prise en compte directe de la forme exacte de l'agent dans l'espace physique est évitée.

Trois sous espaces sont généralement considérés :

- *l'espace libre*, c'est-à-dire l'ensemble des configurations dans lesquelles l'agent peut être sans entrer en collision avec un obstacle.
- *l'espace de contact*, c'est-à-dire l'espace libre avec en plus les configurations de l'agent le mettant en simple contact avec les obstacles. Cet espace est particulièrement utile lorsque l'agent utilise des capteurs tactiles, ou lorsqu'il doit déplacer des obstacles (appuyer sur un interrupteur par exemple).
- *l'espace des obstacles*, il s'agit de toutes les configurations qui correspondent à une position de l'agent dans l'espace physique occupée par un obstacle. C'est donc l'ensemble des configurations causant une collision entre l'agent et un obstacle.

La détermination de l'espace des obstacles est très importante pour planifier des mouvements sans collision. Nous ne faisons ici que citer les méthodes utilisées, une description plus détaillée ainsi qu'une liste de références s'y rapportant se trouve dans [34, p.228-230] :

- Evaluation point par point (*point evaluation*)
- Différence ensembliste de Minkowski (*Minkowski set difference*)
- Méthode des équations de frontière (*boundary equation method*)
- méthode de l'aiguille (*needle method*)
- méthode des volumes de glisse (*sweep volume method*)
- méthode des modèles (*template method*)
- méthode du Jacobien (*Jacobian based method*)

B.3 Planification en *boucle ouverte*

Les méthodes en boucle ouverte présentées ici forment la base des méthodes de planification de mouvements. Sauf mention contraire, elles travaillent dans *l'espace des configurations* et y recherchent un chemin entre la configuration initiale et la configuration finale. Des descriptions plus ou moins approfondies de chacune de ces approches, accompagnées de références vers des publications spécialisées se trouvent dans [43] - **la** référence en matière de planification de mouvements- [34] - un état de l'art du domaine plus concis - et [68, p. 796-806] dont la présentation est plus superficielle mais fait preuve d'une grande clarté et a l'avantage d'être plus récente.

B.3.1 Décomposition cellulaire

L'espace libre est décomposé en différentes cellules et le graphe d'adjacence correspondant est construit. La planification se déroule alors en trois étapes :

1. Identifier la cellule initiale (contenant la configuration initiale) et la cellule finale (contenant la configuration finale).
2. Chercher dans le graphe d'adjacence un chemin reliant les cellules initiale et finale. Soit S la séquence de cellules correspondante.
3. Pour chaque cellule de S calculer un chemin reliant sa bordure avec la cellule précédente à sa bordure avec la cellule suivante. Pour la cellule initiale (resp. finale), le chemin calculé relie la configuration initiale (resp. la bordure de la cellule précédente) à la bordure de la cellule suivante (resp. la configuration finale).

La dernière étape implique que les cellules soient construites de telle manière qu'il soit facile de calculer ces chemins intra-cellulaires. Par exemple, dans une cellule convexe, deux points peuvent toujours être reliés par une ligne droite.

La principale difficulté de cette méthode réside dans la décomposition de l'espace libre, lequel peut avoir une géométrie assez complexe (contenant des courbes en particulier). Idéalement, l'ensemble des cellules le recouvre **exactement** (*exact cell decomposition*) mais la construction et la représentation du découpage correspondant peuvent alors s'avérer très complexes.

En conséquence, une autre approche consiste à construire un découpage recouvrant **approximativement** l'espace libre. Cette approximation se doit d'être conservative sinon l'algorithme présenté ne peut plus garantir que le chemin calculé évite les collisions. videmment, ceci conduit à un certain "gaspillage" de l'espace libre. L'algorithme est alors incomplet : il est possible qu'il ne trouve pas de solution alors qu'il existe effectivement un chemin qui passe par ces parties non recouvertes de l'espace libre.

videmment, plus la décomposition sera fine plus le taux de couverture sera élevé, et plus l'algorithme sera proche de la complétude. En contrepartie, les cellules seront plus nombreuses (parce que plus petites), et la complexité de l'algorithme s'en ressentira. Il s'agit d'un équilibre auquel il faut faire attention.

Une variante permettant d'assouplir ce problème est l'utilisation de cellules de tailles variables. L'idée est de choisir pour chaque cellule la taille maximale compte tenu des obstacles environnant. Les cellules seront petites et nombreuses seulement dans les parties de l'espace libre où sa géométrie est complexe. Le *quadtree* [1] est un exemple de décomposition de ce type.

B.3.2 Rétraction (*Skeletonization*)

Cette approche est aussi appelée la méthode des *autoroutes* ou de la *carte routière* (*road-map approach*).

Alors que la décomposition cellulaire discrétise l'espace libre en une série de parcelles, cette approche le réduit à un ensemble fini de points connectés entre eux, son "squelette" en quelque sorte. C'est une sorte de métaphore du réseau ferroviaire : la planification se déroule en trois étapes :

1. Déterminer les gares les plus proches des configurations initiale et finale.
2. Rechercher dans le graphe des voies ferrées un chemin entre ces gares.

1. L'espace des configuration est d'abord recouvert d'une grille de très gros maillage. Si une cellule n'est pas totalement contenu par l'espace libre, elle est divisée en plus petites cellules. Le procédé continu récursivement jusqu'à l'atteinte d'une taille limite.

3. Calculer un chemin pour relier ces gares aux positions initiale et finale.

La dernière étape est cruciale et doit pouvoir être réalisée facilement, ce qui contraint la construction du réseau.

Parmi les "squelettes" les plus connus, nous pouvons citer :

- Le graphe de visibilité. Ses sommets sont les sommets des obstacles ainsi que les configurations initiale et finale. Un sommet est connecté à tous les sommets visibles depuis sa position. Le chemin le plus court est forcément compris dans le graphe. Il est plutôt adapté aux espaces à deux dimensions avec des obstacles polygonaux.
- Le diagramme de Voronoi. Chacun de ses sommets est situé à égale distance des obstacles les plus proches. Il permet de calculer les chemins les moins dangereux en termes de collision de par la distance aux obstacles qu'ils maintiennent. Lui aussi est plutôt adapté aux espaces à deux dimensions avec des obstacles polygonaux.
- la méthode de la silhouette. Elle peut être utilisée dans un espace de dimension arbitraire mais elle est plus complexe à mettre en œuvre. Brièvement, le principe est de projeter récursivement l'espace des configurations de dimension n dans des espaces de dimension $n-1$. A chaque étape un certain nombre d'éléments sont retenus - ce sont eux qui sont projetés ensuite. Pour fixer les idées, un exemple de résultat est le dessin obtenu lorqu'un tuyau est dessiné en perspective sur une feuille de papier de faon "fil de fer" : deux segments reliés par deux ellipses.

B.3.3 Prise en compte de la dynamique de l'environnement

Une première faon de planifier dans un environnement dynamique est de rajouter une dimension à l'espace des configurations, le temps, et d'utiliser une méthode classique. Bien sûr, le problème devient alors tellement complexe que ces méthodes se montrent peu performantes.

Afin de réduire la complexité, une solution est de ne pas chercher à résoudre directement le problème global - dont le temps est une dimension à part entière, mais plutôt de chercher une solution au problème statique puis de la modifier pour tenir compte des contraintes temporelles. Voici une première faon de procéder :

1. En considérant l'environnement comme statique, trouver un chemin entre les configurations initiale et finale.
2. Construire un espace à deux dimensions : le temps d'une part, et la distance le long du chemin d'autre part. Il est alors possible de représenter les parties du chemin occupées par des obstacles en fonction du temps (le mouvement des obstacles doit être connu a priori).
3. Construire dans cet espace le graphe de visibilité en supprimant les segments que la vitesse maximale de l'agent ne permet pas d'utiliser. Extraire du graphe la trajectoire finale.

Ici, l'influence du temps sur la complexité a été réduite en n'utilisant la dimension temporelle que sur une sous-partie de l'espace des configurations. Une autre approche est de conserver l'espace des configurations dans son intégralité mais de ne prendre en compte qu'une partie de la dimension temporelle. Le chemin calculé n'est alors valide que sur l'intervalle de temps considéré.

1. Les positions successives qu'occupera chaque obstacle dans l'intervalle de temps Δt à venir sont considérées comme des zones interdites dans l'espace de configurations.

En fait, cela revient à projeter dans l'espace de configurations courant les espaces de configurations statiques se succédant pendant Δt.

2. Un chemin est extrait de l'espace des configurations obtenu (en utilisant le graphe de visibilité par exemple).
3. A l'instant $t + \Delta t$, le mouvement des obstacles est réévalué et l'algorithme retourne à l'étape 1.

L'avantage de cet algorithme est que le mouvement des obstacles peut ne pas être connu a priori. Dans ce cas il faut qu'il soit évalué tous les Δt - entre deux replanifications. Il s'agit alors de planification en ligne.

B.4 Planification réactive

B.4.1 Champs de potentiels

La navigation par champs de potentiels est simplissime. Elle a été introduite pour la première fois par [39].

Un champs de potentiels attractif est associé à l'objectif. A chaque point de l'espace est associée une valeur (un *potentiel*) de telle sorte que la topologie de l'ensemble forme un puit centré sur l'objectif : en tout point la pente maximale est inversement proportionnelle à la distance de l'objectif et elle est orientée vers ce dernier. Dans un environnement sans obstacle, il suffirait de "se laisser glisser" le long de la pente du champs de potentiels pour atteindre l'objectif.

Malheureusement, en présence d'obstacles, cette méthode risque de nous mener à une collision (il suffit qu'un obstacle se situe entre la position de départ et l'objectif). Pour éviter cela, un champs de potentiels répulsif est centré sur chaque obstacle. Les obstacles sont alors autant de montagnes.

Les champs de potentiels sont superposés. Le champs global présente alors un minimum à la position de l'objectif et des maxima là où se trouvent les obstacles. La navigation de l'agent revient alors à suivre en tout point la direction de la pente la plus forte, afin de rejoindre l'objectif tout en évitant les obstacles.

Cette méthode est très séduisante car elle permet à l'agent de raisonner localement et donc de manière très rapide. En particulier, la position des obstacles peut n'être connue qu'au moment de l'exécution. En effet, le champs induit par les obstacles peut être calculé en temps réel au fur et à mesure que les obstacles entrent dans le champs des proximètres de l'agent. Par contre, la position relative de l'objectif doit pouvoir être évaluée à tout moment. Ici, le plan découle d'un comportement réactif aux changements d'orientation du gradient du champs global :

- la politique est de suivre la direction opposée à celle du gradient du champs global.
- le champs global (et donc son gradient) peut être partiellement observé lors de l'exécution.

Un avantage de cette politique est qu'elle assure la sécurité de l'agent. En effet, en présence des champs répulsifs uniquement, l'agent suit le diagramme de Voronoi. Dès lors, l'agent ne peut pas entrer en collision.

Malheureusement, la topologie du champs global présente des minima locaux, des sortes de cuvettes, en somme. Si l'agent entre dans l'une d'entre elles il ne peut plus en sortir. Ceci est la contrepartie inévitable du traitement local.

B.4.2 Approche de la *La bande élastique*

Le concept de *bande élastique* est présenté pour la première fois dans [63]. Il s'agit d'un effort pour relier la planification globale, en boucle ouverte, à des traitements locaux améliorant l'exécution du plan, basés sur une approche par champs de potentiels.

L'algorithme est très simple et se déroule comme suit :

1. Un planificateur produit un plan en boucle ouverte : un chemin menant de la configuration initiale à la configuration finale, compte tenu des connaissances a priori sur l'environnement.
2. Le chemin est discrétisé en un ensemble de particules. Ces particules subissent trois forces :
   - Une force de *contraction* qui simule la tension d'un élastique étiré. Elle permet d'obtenir un chemin plus tendu, écartant les éventuelles zone "laches" du chemin initial.
   - Une force *contraignante* qui empêche les particules de se déplacer le long de l'élastique.
   - Une force de *répulsion* qui émane des obstacles et permet d'assurer que le chemin évite les collisions.
3. Lors de l'exécution, la force répulsive est remise à jour en fonction des observations de l'agent. Ainsi, l'élastique (donc le chemin) se déforme pour tenir compte de la position réelle des obstacles, ou même de leurs éventuels mouvements. Si les déformations sont telles que l'élastique "casse", alors cela signifie que la réalité est trop éloignée de ce qui était prévu. Dans ce cas une replanification est lancée.

Afin de diminuer le poids de la mise à jour de la valeur des forces appliquées à chaque particule, Quinlan et al. ont utilisé la notion de *bulle*. Les bulles remplacent les particules dans la discrétisation de la bande élastique. Une bulle est de taille variable : son rayon est égal à la distance de l'obstacle le plus proche, de sorte qu'une bulle occupe toujours le plus d'espace possible. La cohésion de l'élastique est assuré en forcant les bulles voisines à se recouper un peu - quitte à créer ou supprimer certaines d'entre-elles suivant les variations de taille qui se produisent. L'algorithme que nous avons décrit peut être appliqué comme avant, mais cette fois sur moins d'éléments.

Cet algorithme rejoint directement le soucis de n'observer dans l'environnement que ce qui a influencé les décisions du planificateur. En particulier, l'utilisation des bulles permet de ne pas se poser trop de questions quand les obstacles sont lointains. L'intensité de l'observation et du raisonnement lors de l'exécution est directement liée à la réalité de la complexité de l'environnement.

Bibliographie

[1] *Proc. of the 1993 AAAI Spring Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, Stanford, USA.

[2] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows.* Prentice-Hall, 1993.

[3] A. Barrett and D. Weld. Partial-order planning. *Artificial Intelligence Journal*, (67):71–112, 1994.

[4] R. Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, pages 503–516, 1954.

[5] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proc. IJCAI-95*, 1995.

[6] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial Intelligence Journal*, (90):281–300, 1997.

[7] A. Blum and J. Langford. Probabilistic planning in the graphplan framework. In *AIPS Workshop on 'Planning as Combinatorial Search'*. 1998.

[8] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proc. ECP'99*, 1999.

[9] B. Bonet and H. Geffner. Heuristic search planner 2.0. *Artificial Intelligence Magazine*, 2001.

[10] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence Journal*, 129, 2001.

[11] C. Boutilier, R.I. Brafman, and C. Geib. Structured reachability analysis for markov decision processes. In *Proc. UAI'98*, pages 24–32, Madison, jul. 1998.

[12] A. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Brown University, 1998.

[13] E. Codd. A relational model for large shared data banks. *CACM*, 1970.

[14] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proc. AIPS'94*, 1994.

[15] A. Drogoul and D. Servat. Informatique diffuse : vers un monde fait d'agents? In *Proc. JFIADSMA'01*, 2001.

[16] D. Dubois and H. Prade. *Possibility theory - an approach to computerized processing of uncertainty.* Plenum Press, 1988.

[17] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems 2nd Edition.* Addison Wesley, 1994.

[18] P. Fabiani. *Représentation dynamique de l'incertain et stratégie de prise d'information pour un système autonome en environnement évolutif.* PhD thesis, ENSAE, 1996.

[19] P. Fabiani, H.-H. Gonzelez-Banos, J.-C. Latombe, and D. Lin. Tracking an unpredictable target among occluding obstacles under localization uncertainties. *Robotics and Autonomous Systems*, 38(1):31–48, jan 2002.

[20] P. Fabiani and J.-C. Latombe. Dealing with geometric constraints in game-theoretic planning. In *Proc. IJCAI'99*, Stockholm, aug. 1999.

[21] P. Fabiani and Y. Meiller. Planning with tokens. In *Proc. PuK'2000*, Berlin, 2000.

[22] P. Fabiani and Y. Meiller. Théorie des jeux et planification pour le dilemme perception-action. In *Proc. RFIA'2000*, Paris, 2000. AFRIF-AFIA.

[23] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence Journal*, pages 189–208, 1971.

[24] M. Fox and D. Long. Utilizing automatically inferred invariants in graph construction and search. In *Proc. AIPS'2000*, Breckenridge, Colorado - USA.

[25] M. Fox and D. Long. Pddl2.1 : An extension to pddl for expressing temporal planning domains. Technical report, University of Durham, 2001.

[26] M. Friedman and D. Weld. Least-commitment action selection. In *Proc. AIPS'96*, 1996.

[27] F. Garcia. Planification tolérante aux échecs pour un robot mobile. Technical Report DRET:3/89-1534 DERA, ONERA-CERT, mai 1991.

[28] F. Garcia. *Révision des croyances et révision du raisonnement pour la planification.* PhD thesis, ENSAE, 1993.

[29] E. Guere and R. Alami. A possibilistic planner that deals with non-determinism and contingency. In *Proc. IJCAI'99*, 1999.

[30] J. Hoffmann. FF: The fast-forward planning system. *AI magazine*, 2001.

[31] J. Hoffmann. Local search topology in planning benchmarks: A theoretical analysis. In *Proc. AIPS'02*, 2002.

[32] J. Hoffmann and H. Geffner. Branching matters: Altenative branching in graphplan. In *Proc. ICAPS'2003*.

[33] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[34] Y.K. Hwang and N. Ahuja. Gross motion planning - a survey. *ACM Computing Surveys*, 1992.

[35] E. Jacopin. On the foundations of classical planning: Some algebraic and topological structures. In *Proc. PLANSIG'95*, Colchester, UK.

[36] S. Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, pages 67–98, 1997.

[37] S. Kambhampati. Planning graph as (dynamic) csp: Exploiting ebl, ddb and other csp techniques in graphplan. In *Proc. IJCAI'99*, 1999.

[38] S. Kambhampati, E. Lambrecht, and E. Parker. Understanding and extending graphplan. In *Proc. ECP'97*, 1997.

[39] O. Khatib, R. Mampey, and M. Libre. Fonction decision-commande d'un robot manipulateur. Technical report, CERT/DERA, Toulouse, France, 1978.

[40] R. Korf. Linear-space best first search. *Artificial Intelligence Journal*, (62):41–78, 1993.

[41] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proc. AAAI'94*, 1994.

[42] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence Journal*, 1994.

[43] J.-C. Latombe. *Robot Motion Planning*. Kluver Academic Publishers, 1991.

[44] D. Long and M. Fox. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*, pages 87–115, 1999.

[45] D. Long and M. Fox. Extracting route-planning: Firest steps in automatic problem decomposition. In *Proc. AIPS'2000*, Breckenridge, Colorado, USA, 2000.

[46] D. Long and M. Fox. Recognizing and exploiting generic types in planning domains. In *Proc. AIPS'2000*, Breckenridge, Colorado, USA, 2000.

[47] D. Long, M. Fox, L. Sebastia, and A. Coddington. An examination of ressources in planning. In *Proc. Plansig'2000*.

[48] T. Lozano-Perez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):pp. 560–570, 1979.

[49] R. D. Luce and H. Raiffa. *Games and Decisions*. Wiley, New York, 1957.

[50] D. McAllester and D. Rosenblitt. Systematic non linear planning. In *Proc. 9th National Conference on Planning*, 1991.

[51] D. McDermott and AIPS-98 Planning Competition Committee. *PDDL -The Planning Domain Definition Language Version 1.2*, 1998.

[52] Y. Meiller. Planification de déplacements et de prises d'information pour la poursuite d'objets mobiles en présence d'incertitudes. Technical report, ONERA - DCSD, 1999. Rapport d'avancement.

[53] Y. Meiller and P. Fabiani. Perception-action dilemma at planning time : Getting the best out of game theory and classical planning. In *Proc. PLANSIG'99*, 1999.

[54] Y. Meiller and P. Fabiani. Planning with petri nets. In *Proc. RJCIA'00*, 2000.

[55] Y. Meiller and P. Fabiani. Tokenplan ; a planner for both satisfaction and optimization problems. *AI Magazine*, 2001.

[56] M.Gondran and M. Minoux. *Graphes et Algorithmes*. Collection de la Direction des Etudes et Recherches d'Electricité de France, 1990.

[57] Z. Nguyen, R. Sanchez Nigenda, and S. Kambhampati. Altalt: Combining the advantages of graphplan and heuristic state search. Technical report, Arizona State University, 2000.

[58] R. Nigenda, X. Nguyen, and S. Kambhampati. Altalt: Combining the advantages of graphplan and heuristic state search. In *Proc. Knowledge-Based Computer Systems*, Bombay, India, 2000.

[59] J. Penberthy and D. Weld. Ucpop: A sound, complete, partial order planner for adl. In *Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning*, 1992.

[60] M. Peot and D. Smith. Conditional nonlinear planning. In *Proc. AIPS'92*, 1992.

[61] G. Peterson and D. J. Cook. Decision-theoretic planning in the graphplan framework. In *Proc. Workshop on Decision-Theoretic Planning*, Breckenridge - Co, USA, 2000. AIPS'2000.

[62] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Willey & sons, 1994.

[63] S. Quinlan and O. Khatib. Elastic bands : Connecting path planning and robot control. In *Proc. IEEE International Conference on Robotics and Automation*, volume 2, pages 802–807, Atlanta -Georgia, 1993.

[64] I. Refanidis and I. Vlahavas. Grt: A domain independent heuristic for STRIPS worlds based on greedy regression tables. In *Proc. ECP'99*, Durham, UK, 1999.

[65] I. Refanidis and I. Vlahavas. A heuristics based approach to planning in STRIPS domains. *World Scientific*, 2000.

[66] I. Refanidis and I. Vlahavas. The grt planner: Backward heuristic construction in forward state-space planning. *Journal of Artificial Intelligence Research*, (15):115–161, 2001.

[67] L. Siklossy and E. Tulp. Searching time-table networks. *AIEDAM*, 5(3):189–198, 1991.

[68] S.Russel and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice-Hall, 1995.

[69] L. A. Stein. Post-modular systems : Achitectural principles for cognitive robotics. *Cybernetics and Systems*, 28(6):471–487, 1997.

[70] F. Viceriat. Contribution à la comparaison de méthodes de décision. Technical report, ONERA Centre de Toulouse, 2001. Mémoire de stage de deuxième année de l'Ecole des Mines de Saint-Etienne.

[71] V. Vidal. *Recherche dans les graphes de planification, satisfiabilité, et stratégies de moindre engagement*. PhD thesis, Université Paul Sabatier - Toulouse III, 2001.

[72] D. Weld. An introduction to least commitment planning. *AI Magazine*, 1994.

[73] D. Weld. Recent advances in ai planning. *AI Magazine*, 1999.

[74] D. Weld, A. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proc. AAAI'98*, 1998.

[75] D. Weld and D. Smith. Conformant graphplan. In *Proc. AAAI'98*, 1998.

[76] Y. Zemali and P. Fabiani. Optimal heuristic planning using search space splitting : first results (draft). non publié, 2002.

[77] Y. Zemali, P. Fabiani, and M. Ghallab. Using state space splitting to compute heuristics in planning. In *Proc. PLANSIG'01*, 2001.

Du Morcellement de l'espace de recherche en planification d'actions

Diverses approches en planification d'actions consistent à produire un plan solution après avoir construit à partir du problème de planification, puis parcouru, un espace de recherche plus ou moins morcelé en sous-espaces.

En formalisant cette notion de morcellement de l'espace de recherche, nous montrons que des approches en apparence très différentes ne se distinguent en fait que du morcellement qu'elles mettent en œuvre. Nous montrons aussi que la diversité des morcellements est largement sous-exploitée par les planificateurs actuels, ouvrant ainsi la voie à la conception de nouvelles approches, en créant de nouveaux morcellements de l'espace de recherche.

Pour valider expérimentalement notre propos, nous avons développé un planificateur : `TokenPlan`. Nous avons aussi conçu et expérimenté un nouveau morcellement de l'espace de recherche qui permet d'obtenir une approche hybride entre un algorithme *Branch and Bound* et un planificateur disjonctif.

Splitting of the Search Space in Action Planning

Various action planning approaches lead to output a solution plan after the building and exploring of a search space that can be more or less split into sub-spaces.

Formalizing the idea of such a splitting of the search space, we show that apparently very different approaches only display different ways of splitting their search space. In addition, we show that current planners take advantage of few different ways of splitting the search space in comparison with the vast possibilities. This highlights opportunities to create new approaches, through the design of new ways of splitting the search space.

In order to carry out an experimental validation of our work, we developed a new planner named `TokenPlan`. We also designed and used a new way of splitting the search space. It makes it possible to take advantage of both a *Branch and Bound* algorithm and disjunctive planning.

Mots-clés : Intelligence Artificielle, Planification d'actions, Optimisation, Incertitudes, Représentation de l'information, Espace de recherche, TokenPlan, Outils décisionnels.