Contributions à la planification automatique pour la conduite de systèmes autonomes

Damien Pellier

UNIVERSITÉ GRENOBLE ALPES

HABILITATION À DIRIGER DES RECHERCHES

Contributions à la planification automatique pour la conduite de systèmes autonomes

---

Damien PELLIER

Université Grenoble Alpes

Ecole Doctorale Mathématiques, Sciences
et Technologies de l'Information, Informatique

Soutenue publiquement le 5 avril 2018 devant le jury composé de :

Mme Amal El Fallah Professeur Univ. Pierre et Marie Curie ..... Rapporteur
M. Dominique Duhaut Professeur Univ. Bretagne Sud ..... Rapporteur
M. Bernard Moulin Professeur Univ. Laval ..... Rapporteur
Mme Sylvie Pesty Professeur Univ. Grenoble Alpes ..... Examinateur
M. Nicolas Sabouret Professeur Univ. Orsay ..... Examinateur

Remerciements

Je tiens à remercier tout d'abord Bernard Moulin, Dominique Duhaut ainsi qu'Amal El Fallah d'avoir accepté de relire ce mémoire et d'en être rapporteurs. Je remercie également les autres membres du jury Sylvie Pesty et Nicolas Sabouret d'avoir accepté d'assister à la présentation de ce travail.

Je voudrais également remercier tous mes collègues avec qui j'ai travaillé et sans qui ce travail n'aurait pas été possible, en particulier : Humber Fiorino, Bruno Bouzy, Marc Métivier, David Janiszek et encore une fois Sylvie Pesty pour son soutien sans faille.

Enfin, je remercie ma famille pour leur soutien constant et une pensée spéciale pour la relectrice acharnée de ce manuscrit qui se reconnaîtra.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

La conception de systèmes autonomes ou d'agent capables de percevoir, d'apprendre et d'agir sans aucune intervention humaine pour réaliser des tâches complexes est au cœur des recherches en intelligence artificielle depuis de nombreuses années. Les premiers secteurs ayant bénéficié des retombées de ces recherches ont été l'aérospatial, e.g, (FISHER et al., 2000 ; MILLER, 1996) et la défense, e.g, (BERNARDINI, FOX et LONG, 2014). Dans les années à venir ces systèmes autonomes seront de plus en prégnants dans tous les secteurs de l'économie, en passant de l'industrie 2.0 et de l'automatisation de la production manufacturière, à la logistique et aux services de livraisons à domicile (e.g., les drones livreurs développés par Amazon) ou encore aux véhicules sans pilote. L'essor fulgurant de ce dernier secteur est prototypique. En 2004, la DARPA (*Defense Advanced Research Projects Agency*) organisait pour la première fois le DARPA Grand Challenge. Ce challenge consistait à concevoir un véhicule terrestre sans pilote et autonome capable d'effectuer un circuit de 150 milles dans le désert des Mojaves en Californie. Aucune des équipes engagées alors ne parvint à terminer le challenge. En 2007, lors de la troisième édition, six équipes terminèrent le challenge qui fut de plus organisé pour la première fois dans un environnement urbain complexe dans lequel les véhicules engagés étaient soumis aux règles de circulation.

Nous percevons au travers de ces quelques exemples que l'une des clés de l'essor des systèmes autonomes repose sur leur capacité à prendre les bonnes décisions au bon moment pour accomplir les tâches et les objectifs qui leurs ont été confiés. L'essor des systèmes autonomes passe donc par le développement de leur autonomie décisionnelle, i.e., d'algorithmes efficaces leur permettant de décider. Une des branches de l'intelligence artificielle qui s'intéresse à cette problématique est la *planification automatique* (GHALLAB, NAU et TRAVERSO, 2004). C'est dans ce cadre que se situent les travaux que nous présentons dans ce manuscrit. Nos travaux ont pour objectif de développer des méthodes et des techniques génériques et indépendantes du domaine afin de concevoir des systèmes autonomes capables de décider quelles actions réaliser et comment les exécuter pour atteindre un objectif défini *a priori*.

La suite de cette introduction présente succinctement les concepts de la planification automatique et propose un rapide survol des techniques utilisées ainsi que les notations que nous utiliserons tout au long de ce manuscrit. Puis nous abordons plus en détail les principales contributions de nos recherches dans le domaine de la planification automatique pour la conduite de systèmes autonomes réalisées au sein du Laboratoire d'Informatique de Paris Descartes ainsi qu'au sein du laboratoire d'Informatique de Grenoble. Ces contributions sont regroupées en quatre parties. Une première partie traite de nos contributions pour aborder la problématique de la planification en *boucle fermée*, i.e., quand planification et exécution sont entrelacées. Nous abordons notamment le problème de la planification en temps réel et le problème de la planification lorsque le but évolue au cours du temps. Dans

une seconde partie, nous présentons plusieurs de nos contributions dans le cadre de la planification distribuée ou comment un groupe de systèmes autonomes est capable de raisonner conjointement pour élaborer un plan d'actions pour atteindre un objectif partagé ? Dans une troisième partie, nous présentons deux de nos contributions en matière d'heuristique pour la planification automatique. Le développement d'heuristiques performantes est un enjeu majeur pour la planification. C'est l'une des clés permettant le passage à l'échelle des techniques de planification. Finalement, une quatrième partie plus technique présente notre contribution logicielle en termes d'outils pour la planification automatique. L'objectif de ces outils est de faciliter le développement de nouvelles techniques de planification, mais aussi de diffuser plus largement les avancées réalisées dans le domaine. La conclusion de ce manuscrit présentera notre vision des verrous et des enjeux du domaine de la planification pour les années à venir.

1.1 La planification automatique : intuitions et concepts

L'objectif de la planification automatique (FIKES et NILSSON, 1971 ; SACERDOTI, 1975) est de développer des solveurs, aussi appelés planificateurs, capables de générer un plan d'actions à un niveau symbolique à partir d'un état initial pour atteindre un but défini *a priori* indépendamment du domaine d'application. La spécification du problème à résoudre s'appuie généralement sur un langage de haut niveau tel que le langage PDDL (*Planning Domain Description Language*) (GHALLAB et al., 1998). L'algorithme de recherche embarqué dans un planificateur n'est donc pas spécifique au problème à résoudre. L'efficacité du processus de planification dépend des techniques utilisées pour extraire des informations du problème servant à guider la recherche d'un plan solution.

La description d'un problème de planification est réalisée de manière déclarative. En d'autres termes, un problème spécifie ce qu'il faut faire plutôt que comment le faire. La spécification d'un problème de planification, quel que soit le langage utilisé passe par la description de trois éléments : *l'état initial du monde* (les objets à considérer ainsi que leurs propriétés), un *but* et l'ensemble *des actions* qui peuvent être exécutées pour atteindre le but spécifié. Les actions sont définies en termes de *préconditions* et d'*effets*. Les préconditions expriment les propriétés du monde qui doivent être vérifiées pour pouvoir appliquer l'action et les effets expriment les conséquences de l'exécution de l'action sur le monde. L'exécution d'une action fait donc évoluer l'état du monde. Comme certaines actions ne sont pas réversibles ou qu'elles produisent des effets contradictoires, l'ordre d'exécution des actions est important. La solution d'un problème de planification est dans le cas général un ensemble ordonné d'actions, appelé plan, qui peut s'exécuter à partir de l'état initial du problème et qui produira un état du monde vérifiant le but.

Un exemple classique de problème de planification est celui de la logistique. Une description de ce problème dans le langage PDDL est donnée à la section 10.1. Les objets du monde sont des paquets, des camions, des avions, des aéroports et des emplacements. Les paquets peuvent être chargés et déchargés de camions et d'avions. Les avions et les camions peuvent se déplacer d'aéroport en aéroport et de ville en ville. Le chargement, le déchargement ainsi que les déplacements constituent les actions du problème. L'état initial définit l'emplacement initial des paquets, des avions et des camions. Finalement, le but spécifie la destination finale des paquets. Un plan

solution dans notre exemple de logistique serait une séquence d'actions de chargements, de déchargements et de déplacements de véhicules permettant de déplacer tous les paquets à leur lieu de livraison.

Plus formellement, la résolution d'un problème de planification s'exprime comme la recherche d'un chemin représentant un plan d'actions dans un système à base de transitions partant d'un état initial en choisissant quelles actions doivent être appliquées pour atteindre un objectif donné. L'objectif peut être formalisé comme un état but, représenté par un ensemble de propositions logiques, une fonction d'utilité associée aux états, devant être minimisée ou maximisée, ou encore une tâche complexe devant être décomposée.

Définition 1.1 (Système à base de transitions)**.** *Un système à base de transitions est un tuple* $\Sigma = (S, A, c, \gamma)$ *tel que :*

— S *est un ensemble d'états ;*
— A *est un ensemble d'actions ;*
— $c : A \to \mathbb{R}^+$ *est une fonction de coût définie pour chaque action ;*
— $\gamma : S \times A \to S$ *est une fonction de transition.*

Un système à base de transitions se représente par un graphe orienté dont les nœuds sont des états S et les arcs les actions de A. Étant donnés une action a et un état s, si la fonction de transition $\gamma(s, a)$ est définie, alors l'action a est *applicable* dans l'état s. Appliquée a dans l'état s produit un nouvel état $s' = \gamma(s, a)$ avec un coût de $c(a)$. Σ est déterministe si pour tous les états s et toutes les actions a, la fonction de transition $\gamma(a, s)$ produit un unique état résultat s'. Σ a un coût constant si $a \in A$, $c(a) = 1$. Un plan est une séquence d'actions $\pi = \langle a_1, \cdots, a_k \rangle$ de longueur k. L'état produit par l'application de π dans un état s et l'état obtenu par l'application séquentielle de chaque action du plan π. Nous notons par extension l'état résultant de l'exécution d'un plan π comme suit :

$$\gamma(s, \pi) = \begin{cases} s, & \text{if } k = 0 \\ \gamma(\gamma(s, a_1), \langle a_2, \cdots, a_k \rangle), & \text{if } k > 0 \text{ et } a_1 \text{ est applicable dans } s \\ \text{indéfinie}, & \text{sinon} \end{cases}$$

Un état s' est atteignable à partir d'un état s, s'il existe un plan π tel que $s' = \gamma(s, \pi)$.

1.2 Le modèle conceptuel de la planification

Il est consensuel de présenter la dynamique d'un système autonome planifiant comme l'interaction de trois composants (cf. figure 1.1) : un *système à base de transitions* qui, comme présenté, modélise la dynamique de l'environnement dans lequel évolue le système autonome ; un *contrôleur* qui fournit, étant donné un état et un plan, la meilleure action à effectuer pour réaliser le but fixé ; et finalement un *planificateur* qui, étant donné une description du système, un état initial et un objectif, recherche un plan pour l'atteindre.

Le modèle présenté, bien que didactique, n'est pas opérationnel. Il cache non seulement un certain nombre d'hypothèses implicites, mais également la palette des différentes techniques disponibles pour le rendre opérationnel. Parmi ces hypothèses, nous pouvons citer :

— *la complète observabilité de l'environnement :* le système connaît à tout instant l'état dans lequel il se trouve et peut donc prédire l'état dans lequel il sera

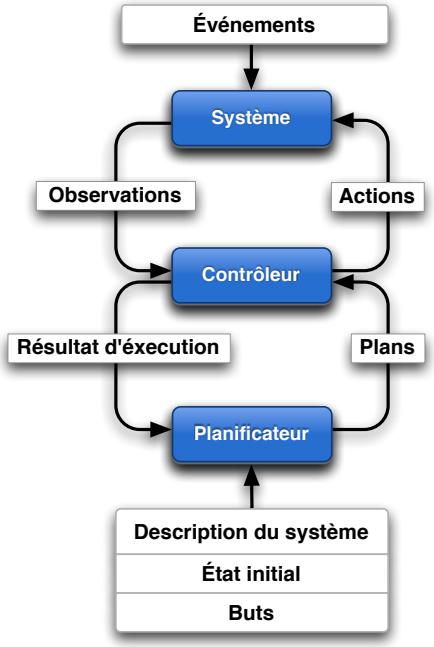


FIGURE 1.1 – Modèle conceptuel de la planification

après l'exécution d'une action. Dans le cas contraire, le système doit être capable de gérer l'ambiguïté de son état en considérant non plus un seul état courant, mais un ensemble d'états courants possibles.

— *le déterminisme de l'environnement :* l'exécution d'une action produit toujours le même état. Si cette hypothèse est levée, le système doit alors être capable de gérer des plans contenant des alternatives, e.g., en utilisant des constructions conditionnelles de la forme *faire a puis en fonction de son résultat faire b ou c*.
— *le caractère statique de l'environnement :* l'environnement ne possède pas de dynamique interne, autrement dit aucun événement ne peut survenir dans le système.
— *la gestion implicite du temps :* les actions n'ont pas de durée, les changements de l'environnement sont instantanés et la concurrence entre les actions n'est pas prise en compte.
— *le processus de planification est hors ligne :* le processus de planification ne prend pas en compte le retour de l'environnement après l'exécution des actions.
— *la centralisation du processus de planification :* la planification se limite à la planification de l'activité d'un seul système autonome.
— etc (GHALLAB, NAU et TRAVERSO, 2004).

1.3 Les grandes techniques de résolution

Depuis maintenant plus de 30 ans, de nombreuses techniques ont été développées pour répondre au problème de la planification et lever les hypothèses précédemment citées. Parmi celles-ci, *les techniques de recherche dans les espaces d'états* sont les plus simples. Récemment, les planificateurs reposant sur cette technique couplée à une recherche en chaînage avant et à une heuristique qui estime la distance au but en ignorant les effets négatifs des actions, ont permis à la planification

d'effectuer un saut quantitatif sur le plan des performances (HOFFMANN et NEBEL, 2001 ; RICHTER et WESTPHAL, 2008). *Les techniques de recherche dans un espace de plans* (PENBERTHY et WELD, 1992) fonctionnent par raffinements successifs d'un plan, jusqu'à ce que tous les conflits entre les actions soient résolus ou que toutes les préconditions des actions soient produites par une autre action du plan ou encore qu'elles soient vérifiées dans l'état initial. *Les techniques utilisant des graphes de planification* (BLUM et FURST, 1997) reposent sur deux idées : une analyse d'atteignabilité et une technique de raffinement disjonctive, qui consiste à adresser un ou plusieurs conflits en utilisant une disjonction de résolveurs. *Les techniques* SAT (*SATisfiability problem*) encodent le problème de planification sous la forme d'un problème de satisfiabilité et effectuent ensuite la recherche d'un plan solution en s'appuyant sur des algorithmes SAT connus (KAUTZ, SELMAN et HOFFMANN, 2006 ; RINTANEN, 2014). Le problème de planification peut être également encodé sous la forme d'un problème CSP (*Constraints Satisfaction Problem*) (BARTÁK et TOROPILA, 2008 ; COOPER, ROQUEMAUREL et RÉGNIER, 2011 ; KAMBHAMPATI, 2000). Pour les deux dernières techniques, l'idée sous-jacente est de bénéficier directement des avancées dans ces deux domaines connexes pour accélérer la synthèse de plans solutions. Les techniques HTN (*Hierarchical Transition Network*) sont similaires aux techniques de recherche dans les espaces d'états dans la mesure où chaque état est représenté par un ensemble de propositions. Toutefois, les planificateurs HTN (EROL, HENDLER et NAU, 1994 ; NAU et al., 2003 ; RAMOUL et al., 2017) diffèrent dans leur façon de chercher un plan puisqu'ils fonctionnent par décomposition récursive d'une tâche complexe en tâches primitives. Avec *les techniques* MDP (*Markov Decision Process*) (MARECKI et TAMBE, 2008 ; SRIDHARAN, WYATT et DEARDEN, 2008), le problème de planification consiste à développer une politique optimale, i.e., à associer à un état une action qui maximise une récompense globale. Cette dernière technique est largement adoptée pour traiter les problèmes de planification non déterministe. Finalement, des *techniques issues du Model Checking* (PISTORE et TRAVERSO, 2001) permettent de planifier en prenant en compte l'incertitude, le non-déterminisme et l'observabilité partielle de l'environnement.

Le panorama des techniques de planification que nous venons de présenter ne se veut pas exhaustif. Bien d'autres techniques existent. Nous verrons dans les différents chapitres de ce manuscrit comment ces techniques peuvent être étendues pour lever certaines hypothèses du modèle classique de planification.

1.4 Représentation classique des problèmes de planification

La communauté planification utilise principalement le langage PDDL pour décrire des problèmes de planification. Toutefois, PDDL n'est pas le seul langage utilisé dans la communauté, e.g., OCL (*Object Constraints, Language*) (LIU et MCCLUSKEY, 2001) ou encore OPL (*Other Planning Language*) (MCDERMOTT, 2005). Le langage PDDL repose sur une représentation logique des problèmes de planification.

Chaque état s du monde est représenté par un ensemble de propositions logiques. Si une proposition p est dans un état s alors p est considérée comme vraie dans l'état s et fausse dans le cas contraire. Chaque proposition logique définit une propriété du monde, e.g., un camion est dans dépôt, un avion est situé dans un aéroport, etc. Une action est une expression qui définit les propositions qui doivent être vérifiées dans un état pour s'appliquer et les propositions à ajouter et à supprimer à l'état courant après l'application de l'action. Dans cette représentation, les

actions modifient la valeur de vérité des propositions, i.e., des propriétés du monde. Par simplicité et par concision, les états et les actions utilisent classiquement la logique du premier ordre (sans symbole de fonction dans les cas simples) plutôt que la logique propositionnelle. On parle alors d'opérateurs de planification.

Définition 1.2 (Opérateur). *Un opérateur o est un tuple $o = (name(o), precond(o), effect(o))$ dont les éléments sont :*
- *$name(o)$, le nom de l'opérateur,*
- *$precond(o)$, les préconditions de l'opérateur, i.e., l'ensemble des littéraux qui expriment les propriétés du monde qui doivent être vérifiées pour appliquer o ;*
- *$effect(o)^-$, l'ensemble des littéraux qui expriment les propriétés du monde qui ne sont plus vérifiées après l'exécution de o ;*
- *$effect(o)^+$, l'ensemble des littéraux qui expriment les propriétés du monde qui sont vérifiées après l'exécution de o.*

On appelle paramètres d'un opérateur l'ensemble des variables utilisées dans ses préconditions et dans ses effets.

Définition 1.3 (Action). *Une action est une instance d'un opérateur, i.e., un opérateur dont tous les paramètres sont instanciés. Si a est une action et s un état tel que les préconditions de a sont vérifiées dans s, alors a est applicable dans s et l'état résultant de l'application de a dans s et l'état s' :*

$$s' = \gamma(s, a) = (s - effects^-(a)) \cup effects^+(a)$$

Définition 1.4 (Un problème de planification). *Un problème de planification est un quintuplet (P, A, c, s_0, g) où :*
- *P est un ensemble fini de propositions ;*
- *A est un ensemble fini d'actions, pour chaque action $a \in A$, $precond(a)$ et $effect(a)$ sont des sous-ensembles de P et $effects^-(a) \cap effects^+(a) = \emptyset$;*
- *$c : A \to \mathbb{R}^+$ est une fonction de coût ;*
- *$s_0 \subseteq P$ est l'état initial ;*
- *$g \subseteq P$ est le but.*

Pour simplifier les notations, nous omettrons de préciser la fonction de coût lorsque celle-ci est uniforme par la suite.

Définition 1.5 (Espace de recherche). *L'espace de recherche d'un problème de planification (P, A, c, s_0, g) dans cette représentation est un système à base de transitions (S, A, c, γ) où $S \subseteq 2^P$ est un sous-ensemble des états définis sur P.*

Définition 1.6 (Plan solution). *La solution d'un problème de planification (P, A, c, s_0, g) est un plan π tel que $g \subseteq \gamma(s_0, \pi)$.*

Le calcul d'un plan solution pour un problème de planification est un problème NP-difficile dans le cas général. Les techniques de planification ne peuvent donc pas reposer sur des algorithmes de recherche n'exploitant que la force brute de calcul des machines. C'est pourquoi la plupart des algorithmes de recherche développés s'appuient sur des techniques de recherche informée et des heuristiques informatives exploitant la structure des problèmes de planification. Nous reviendrons sur ce point dans la partie III de ce manuscrit.

Première partie

Planifier en boucle fermée

DANS des environnements dynamiques, les agents doivent constamment s'adapter pour faire face aux événements qui viennent perturber leurs plans en entrelaçant planification et exécution. On parle alors de *planification en boucle fermée*. Dans cette partie, nous proposons d'aborder ce problème sous deux angles différents.

Tout d'abord, nous nous intéressons à ce problème sous l'angle de *la planification temps réel*. Planifier en temps réel est souvent une nécessité lorsque choisir les actions à exécuter n'est qu'une fonction au sein de l'architecture complexe d'un système autonome où le temps alloué au traitement de chaque fonction est limité. Étant donnée la contrainte temporelle forte, les mécanismes de planification temps réel vont consister à explorer un sous-ensemble de l'espace de recherche autour de l'état courant de la manière la plus intelligente possible en vue d'éclairer au mieux le choix des prochaines actions à exécuter. Ce problème a été peu étudié dans le cadre de la planification de tâches. Choisir en temps réel, i.e., en temps constant inférieur au temps nécessaire pour trouver un plan solution complet, la ou les meilleures actions à réaliser tout en entrelaçant planification et exécution jusqu'à ce que le but soit atteint, soulève de nombreuses difficultés. En particulier, le fait de commencer à exécuter les premières actions d'un plan sans aucune garantie que ces actions permettront d'atteindre le but peut conduire le système dans des états puits non souhaités. De plus sous cette hypothèse, il n'est pas possible de garantir l'optimalité des plans solutions exécutés. Dans ce contexte, nous présentons deux de nos contributions financées par le projet ANR Explo-RA : une première contribution (chapitre 2) s'inspirant des algorithmes de recherche RTS (*Real-Time Search*) (KORF, 1990) couplée avec des techniques heuristiques spécifiques à la planification de tâches et une seconde contribution (chapitre 3) fondée sur les techniques de recherche arborescente de Monte-Carlo MCTS (*Monte Carlo Tree Search*) (CHASLOT et al., 2008). Nous montrons comment nos deux contributions améliorent la qualité des décisions prises en temps réel ainsi que la qualité du plan exécuté.

Pour conclure cette partie, nous abordons le problème de la planification en boucle fermée sous un angle plus dynamique en faisant l'hypothèse que le but peut évoluer pendant la recherche d'un plan solution ou pendant son exécution. En effet, il n'est pas rare que des événements conduisent à une modification du but initial qui a été confié au système. Ce cas se produit notamment lorsque le système est en interaction avec un opérateur humain qui peut venir à tout instant modifier le but précédemment assigné. Dans de tel cas, comment replanifier tout en exploitant au mieux les précédentes recherches afin de limiter l'effort de synthèse d'un nouveau plan pour atteindre le nouvel objectif ? Nous proposons dans le chapitre 4 une contribution inspirée des algorithmes de recherche de cible mouvante MTS (*Moving Target Search*) (ISHIDA et KORF, 1991) pour répondre à cette question.

Chapitre 2

Planifier en temps réel en s'inspirant des algorithmes de recherche temps réel

POUR répondre à la problématique de la planification temps réel, nous proposons dans ce chapitre d'adapter les algorithmes classiques de recherche temps réel RTS (*Real Time Search*), développés principalement dans le contexte des jeux vidéo pour résoudre des problèmes de planification de trajectoires, à la planification de tâches.

L'intérêt des algorithmes RTS (KORF, 1990) réside dans le fait qu'ils sont conçus pour entrelacer planification et exécution en faisant l'hypothèse que le temps alloué à la planification ne permet pas de trouver un plan solution. Sous cette hypothèse, le problème de planification consiste seulement à déterminer la ou les meilleures actions à exécuter en explorant le sous-ensemble de l'espace de recherche autour de l'état courant dans le temps imparti. Par conséquent, le plan solution finalement trouvé et exécuté n'est pas nécessairement optimal et le planificateur peut échouer dans sa recherche d'un plan solution, e.g., en exécutant une action qui le conduit dans un puits de l'espace de recherche. L'exploration du sous-espace de recherche s'effectue généralement avec une simple recherche en largeur d'abord. Ceci laisse une large place à l'amélioration du mécanisme de sélection d'actions notamment en l'adaptant et en le généralisant à la planification de tâches. Dans ce contexte, notre contribution (PELLIER, BOUZY et MÉTIVIER, 2011) consiste à proposer un nouvel algorithme de planification temps réel, appelé LRTP (*Learning Real-Time Planning*), qui intègre deux techniques pour guider de manière intelligente la recherche d'un plan solution. La première technique guide l'exploration du sous-espace de recherche local en s'appuyant sur une technique de planification incrémentale utilisant un agenda de buts. La seconde technique permet au système de choisir plusieurs actions à chaque pas de décision. Dans la suite, nous noterons la première technique (I) pour incrémentale et (J) pour saut.

Ce chapitre est organisé de la manière suivante : la section 1 et 2 donne une vue d'ensemble des travaux dans le domaine des algorithmes de type RTS et des principales techniques d'agenda de buts. La section 3 présente notre contribution et les notations utilisées notamment le pseudo-code de LRTP avec les deux améliorations (I) et (J). La section 4 présente les résultats expérimentaux obtenus par notre approche montrant que l'agenda de buts ainsi que les sauts améliorent de manière significative le processus de sélection d'actions dans le cadre de la planification temps réel.

2.1 Les algorithmes de recherche temps réel

Korf a été le premier à proposer un algorithme capable d'entrelacer planification et exécution. Cet algorithme s'appelle Real-Time A* (RTA*) (KORF, 1990). Il effectue une recherche de type A* en un temps limité. Il a également proposé une version, appelée LRTA*, capable d'apprendre à partir des résolutions répétées d'un même problème la fonction heuristique permettant à LRTA* de converger vers le plan optimal. Chaque cycle de planification et d'exécution est appelé un épisode. De nombreux autres algorithmes RTS ont été proposés, par exemple, l'algorithme SLA* (Search and Learn A*) (SHUE et ZAMANI, 1993). Cet algorithme, présenté comme une amélioration de LRTA*, inclut un mécanisme de *backtracking*. FALCONS (FURCY et KOENIG, 2000), LRTA*$_{LS}$ (HERNÁNDEZ, CONCEPCIÓN et MESEGUER, 2009) ou encore LSS-LRTS* (KOENIG et SUN, 2009) sont également des extensions de LRTA*. Ces algorithmes se focalisent sur les mécanismes d'apprentissage. L'objectif ici est de converger le plus rapidement possible sous certaines hypothèses vers le plan optimal. Nous pouvons encore citer l'algorithme γ-Trap (BULITKO, 2004). La particularité de cet algorithme est de pondérer la fonction heuristique comme le fait l'algorithme weighted-A*. Il est également capable de choisir à chaque étape non pas une seule action, mais une séquence d'actions. Cette amélioration lui permet de faire des sauts dans l'espace de recherche. Enfin, citons les travaux de (BULITKO et LEE, 2006) qui proposent un cadre unificateur pour la recherche temps réel incluant les principales améliorations proposées dans les algorithmes précédemment cités.

2.2 Les techniques de planification incrémentale

Une manière de guider la recherche d'un plan solution consiste à ordonner les propositions du but d'un problème de planification puis à rechercher incrémentalement un plan solution pour atteindre chacune des propositions indépendamment dans l'ordre défini. Cette technique heuristique, appelée planification incrémentale ou dirigée par un agenda de buts, améliore les performances de la recherche en guidant le planificateur vers la construction progressive d'un plan solution. Si un plan pour atteindre une proposition n'invalide pas une proposition ordonnée avant dans l'agenda de buts, cela signifie que l'agenda a découpé le but global en sous-buts pouvant être résolus de manière indépendante. Dans ce cas, le problème de planification initial a été découpé en deux sous-problèmes plus simples et indépendants. Inversement, si plusieurs propositions ordonnées avant sont invalidées, l'ordre fourni par l'agenda de buts n'est pas informatif, car les efforts consentis précédemment pour les atteindre s'avèrent être inutiles. Dans ce cas, les propositions invalidées et la proposition courante sont liées et ne peuvent être résolues indépendamment. Par conséquent, l'efficacité du processus de planification incrémentale dépend fortement de la pertinence de l'ordre entre les propositions qui est calculé, mais aussi de l'interdépendance intrinsèque des propositions du but des problèmes à résoudre.

Koehler et Hoffmann ont été les premiers à essayer de déterminer une relation d'ordre entre les propositions du but d'un problème en introduisant la notion *d'ordre raisonnable* (KOEHLER et HOFFMANN, 2000). Cette notion définit que deux propositions A et B sont raisonnablement ordonnées s'il n'est pas possible d'atteindre un état tel que A et B soient vérifiées, à partir d'un état où seulement B est vérifiée, sans avoir temporairement invalidée B. Dans une telle situation, A doit être raisonnablement ordonnée avant B pour limiter des recherches inutiles. Deux principales

extensions de ce travail préliminaire ont été proposées. La première, appelée *landmarks planning*, a été proposée par (HOFFMANN, PORTEOUS et SEBASTIA, 2004). Cette extension n'ordonne pas seulement le but global, mais également d'autres propositions qui apparaissent nécessairement au cours de la recherche d'un plan solution. Les propositions sont appelées des *landmarks*. La principale caractéristique des *landmarks* est qu'ils doivent être vérifiés à un instant donné, quel que soit le chemin solution emprunté. Cette approche a notamment été implémentée avec succès dans le planificateur LAMA (RICHTER et WESTPHAL, 2008).

La seconde extension a été présentée par (HSU, WAH et CHEN, 2005). La construction de l'agenda de buts peut se résumer en trois étapes :

1. construire un graphe de planification (BLUM et FURST, 1997) à partir de l'état initial jusqu'à ce que le point fixe du graphe soit atteint sans calculer les exclusions mutuelles ;
2. extraire du graphe de planification un plan relaxé en ignorant les effets négatifs des actions de manière identique au calcul de la fonction heuristique réalisé par le planificateur FastForward (HOFFMANN et NEBEL, 2001) ;
3. déterminer un ordre entre chaque paire A et B de propositions en examinant toutes les actions du graphe de planification qui rendent B vraie. Il est important de noter que toutes les relations d'ordre partiel détectées par l'approche de Koehler le sont également par cette approche. Cette approche est donc strictement meilleure que la précédente.

2.3 L'algorithme LRTP

Notre contribution, l'algorithme LRTP (*Learning Real-Time Planning*), s'inscrit dans le cadre de la planification au sens STRIPS (FIKES et NILSSON, 1971) telle que définie §1.4. Dans la suite de ce chapitre, nous nous focalisons uniquement sur la phase de sélection d'actions. Le mécanisme d'apprentissage utilisé dans LRTP est identique à celui proposé par (BULITKO et LEE, 2006). À chaque épisode, la valeur de la fonction heuristique H calculée pour chaque état est conservée. Si pendant une nouvelle recherche un état déjà exploré s lors d'un précédent épisode est à nouveau construit la valeur heuristique de s est mis à jour en appliquant la règle :

$$H(s) = max\{H(s), min_{s' \in \text{successeurs}(s)}\{1 + H(s')\}\} \tag{2.1}$$

Nous supposons ici que chaque action possède un coût constant de 1.

2.3.1 La procédure de planification

Le pseudo-code de LRTP est décrit par l'algorithme 1. Il prend en entrée un problème de planification (A, s_0, g), le temps de décision t_d ainsi que le temps global alloué à la recherche t_g. Dans un premier temps, LRTP exécute une boucle (ligne 1) dans laquelle il exécute itérativement des épisodes tant que le temps global alloué t_g n'est pas dépassé. s_0 représente l'état initial à partir duquel débutent les différents épisodes. s représente l'état courant à partir duquel la phase de planification est initiée, et s_r est l'état courant du monde à partir duquel les actions sont exécutées (ligne 2). Dans un second temps, LRTP entre dans la boucle de décision (ligne 3). À chaque itération, LRTP cherche la meilleure séquence d'actions $\pi' = \langle a_1, \ldots, a_n \rangle$ qui le conduit au but g. La recherche est réalisée par la procédure IASA* avec l'amélioration (I) (ligne 4) et par la procédure ASA* (ligne 5) dans le cas contraire.

Algorithme 1 : LRTP(A, s_0, g, t_d, t_g)

```
1  while elapsed_time < t_g do
2      π ← ⟨⟩, s ← s_0, s_r ← s_0
3      while elapsed_time < t_g and g ⊈ s_r do
4          if I is on then π' ← IASA*(A, s, g, t_d)
5          else π' ← ASA*(A, s, g, t_d)
6          Let π' = ⟨a_1, ..., a_k⟩
7          if J is on then π ← π · π', s ← γ(s, π')
8          else π ← π · a_1, s ← γ(s, ⟨a_1⟩)
9          Execute and remove the first action a of π
10         s_r ← γ(s_r, ⟨a⟩)
```

Lorsque l'amélioration (J) est active (ligne 7), LRTP concatène la séquence d'actions π' au plan courant π et met à jour son état courant s avec la fonction $\gamma(s, \pi')$. Avec cette extension, LRTP saute de son état courant à l'état résultant de l'application de π' qui devient alors le nouvel état courant s. Lorsque l'amélioration (J) est inactive (ligne 8), LRTP concatène seulement la première action a_1 de π' au plan courant π de l'épisode et met à jour s en utilisant la fonction $\gamma(s, \langle a_1 \rangle)$. Par la suite, quel que soit l'état d'activation de l'amélioration (J), LRTP dépile et supprime la première action du plan π et l'exécute. Finalement, s_r est mis à jour (ligne 10).

Lorsque l'amélioration (J) est active, LRTP ne peut pas modifier les actions qui sont stockées dans son tampon. Ceci a pour résultat que s_r et s peuvent être différents. LRTP tire parti du fait que plusieurs actions aient été mises en zone tampon en une seule décision pour accumuler un crédit de temps qu'il utilisera pour sa prochaine décision. Cette technique permet à LRTP d'effectuer une recherche plus approfondie de l'espace de recherche local et d'augmenter ainsi la qualité des séquences d'actions obtenues.

2.3.2 La sélection d'actions

L'objectif de l'étape de la sélection d'actions est de calculer en temps limité la séquence d'actions la plus proche du plan solution optimal pour le problème de planification donné. Pour atteindre cet objectif, nous nous appuyons sur l'algorithme A* (HART, NILSSON et RAPHAEL, 1968). Le pseudo-code décrivant cette procédure, appelée ASA*, est décrit par l'algorithme 2. Cet algorithme de sélection d'actions présente la procédure de sélection non guidée par un agenda de buts. Son principe est simple. Lorsque le temps de décision t_d est écoulé, *ASA** choisit l'état le plus proche du but parmi la liste des états contenus dans la liste S_{open} (ligne 1). La séquence d'actions retournée est le chemin reliant l'état initial s à l'état feuille choisi. La sélection de l'état le plus proche du but dans S_{open} s'effectue en trois étapes. Premièrement, *ASA** sélectionne les états ayant la valeur de f la plus basse (ligne 2). Ces états sont stockés dans S_f. Puis dans un second temps, *ASA** choisit parmi les états de S_f ceux qui possèdent la plus petite valeur de g et les stocke dans S_g (ligne 3). Finalement, *ASA** désigne de manière aléatoire le nouvel état courant s parmi les états S_g (ligne 4). L'idée sous-jacente ici est de donner la priorité aux états les plus proches du but, mais également aux états les plus proches de l'état initial afin d'augmenter la robustesse des séquences d'actions retournées. Cette technique n'est pas nouvelle (BULITKO et LEE, 2006).

Algorithme 2 : ASA*(A, s_0, g, t_d)

1 $S_{open} \leftarrow$ A*(A, s_i, g_i, t_d)
2 Let S_f the set of states with the lowest f-value of S_{open}
3 Let S_g the set of states with the lowest g-value of S_f
4 Choose randomly a state s from S_g
5 **return** the plan from s_0 to s

Le pseudo-code IASA* décrit la procédure de sélection d'actions dirigée par les buts (cf. Algorithme 3). Dans un premier temps, la procédure de sélection calcule l'agenda de buts en s'appuyant sur les travaux de (HSU, WAH et CHEN, 2005), i.e., en déterminant toutes les relations d'ordre entre les propositions atomiques du but (ligne 2). Le but g est dorénavant une séquence de propositions atomiques $\langle g_1, \ldots, g_n \rangle$ telle que $g = \bigcup_{i=1}^{n} g_i$. Puis, pour chaque proposition atomique g_j, et ce tant que le temps alloué à la recherche locale n'est pas dépassé, la procédure de recherche locale *ASA** est lancée pour déterminer un plan ayant pour état initial s_i (initialement assigné à s_0) et permettant d'atteindre incrémentalement le but $g_i = \bigcup_{j=1}^{i} g_j$ (lignes 4-5). Le plan solution pour le sous-but g_i est alors ajouté à la séquence d'actions en cours de construction et l'état s_i est mis à jour en appliquant la fonction $\gamma(s_i, \pi')$ (ligne 7). Finalement, si toutes les propositions atomiques constituant le but sont vérifiées ou si le temps de décision alloué est dépassé, la recherche prend fin et la procédure retourne la séquence d'actions calculée.

Algorithme 3 : IASA*(A, s_0, g, t_d)

1 $\pi \leftarrow \langle\rangle$, $i \leftarrow 1$, $s_i \leftarrow s_0$
2 $\langle g_1, \ldots, g_n \rangle \leftarrow$ RelaxedPlanOrdering(A, s_0, g)
3 **while** $elapsed\_time < t_d$ *and* $i \leq n$ **do**
4 $\quad g_i \leftarrow \bigcup_{j=1}^{i} g_j$
5 $\quad \pi' \leftarrow$ ASA*$(A, s_i, g_i, t_d - elapsed_t ime)$
6 $\quad \pi \leftarrow \pi \cdot \pi'$
7 $\quad s_i \leftarrow \gamma(s_i, \pi')$
8 $\quad i \leftarrow i + 1$
9 **return** π

2.4 Expérimentation et évaluation

L'objectif de ces expériences est d'évaluer les performances de LRTP, avec ou sans l'amélioration (I), et avec ou sans l'amélioration (J), sur des problèmes classiques de planification tirés des compétitions internationales de planification (IPC). Nous utilisons la fonction heuristique non admissible [1] de FastForward (HOFFMANN et NEBEL, 2001) pour conduire la recherche. Nous avons quatre variantes à évaluer : LRTP sans amélioration, LRTP avec planification incrémentale (LRTP+I), LRTP avec sauts (LRTP+J), et LRTP avec les deux améliorations (LRTP+IJ). Les quatre variantes ont été implémentées en Java en s'appuyant sur la bibliothèque PDDL4J [2].

1. Une fonction heuristique qui ne surestime jamais la distance (ou plus généralement le coût) pour atteindre le but est considérée comme admissible.
2. https://github.com/pellierd/pddl4j

2.4.1 Cadre expérimental

Pour nos quatre variantes, nous voulons observer : (1) le pourcentage de réussite, i.e., le pourcentage d'épisodes pour lesquels l'objectif a été atteint et (2) la qualité moyenne (nombre d'actions) des plans solution trouvés sur les épisodes considérés en fonction du temps de décision alloué. Pour évaluer ces deux critères, nous avons mis en place 3 expérimentations :

Expérimentation 1. Les quatre algorithmes sont évalués sur un problème spécifique issu d'IPC-5, le problème 15 du domaine Rovers, choisi pour sa difficulté moyenne et mettre en évidence la façon dont les différents algorithmes se comportent selon le temps de décision. Chaque expérience comprend 10 épisodes. Un épisode se termine sur un succès si l'algorithme atteint l'objectif ou sur échec si l'épisode exécute 400 actions sans atteindre le but.

Expérimention 2. Les quatre algorithmes sont évalués sur tous les problèmes du domaine Rovers avec un temps de décision donné. Chaque expérience comporte 100 épisodes. Les épisodes se terminent si l'algorithme atteint l'objectif ou si 500 actions ont été exécutées sans atteindre le but. Les résultats fournis sont les performances moyennes sur chaque problème du domaine. L'objectif de cette expérience est de montrer que les comportements observés sur un problème spécifique sont généralisables à l'ensemble des problèmes d'un domaine.

Expérimention 3. Nous présentons les résultats obtenus par les quatre algorithmes sur tous les problèmes de 15 domaines extraits des différentes compétitions de planification. Un temps de décision spécifique est utilisé pour chaque domaine. Comme précédemment chaque expérience comporte 100 épisodes et chaque épisode se termine si l'algorithme atteint l'objectif ou si 500 actions ont été exécutées sans atteindre le but. Les résultats fournis sont les performances moyennes sur chaque domaine. Si un algorithme n'a jamais atteint l'objectif pour un problème donné, la longueur de son plan est fixée à 500. L'objectif de cette expérience est de montrer que les comportements observés sur tous les problèmes d'un domaine sont généralisables à plusieurs domaines.

Tous les tests ont été effectués sur un processeur Intel Core 2 Quad 6600 (2,4 GHz) avec 4 Go de mémoire vive.

2.4.2 Expérimentation 1. (Le problème 15 du domaine Rovers)

Ce paragraphe présente les résultats obtenus par les quatre algorithmes pour résoudre le quinzième problème du domaine Rovers (IPC-5) en termes de pourcentage de réussite et de qualité de la sélection d'actions.

Pourcentage de réussite

La figure 2.1 donne le pourcentage de réussite de chaque variante de LRTP pour le problème 15 du domaine Rovers en fonction du temps de décision alloué. LRTP sans amélioration obtient 100% de réussite pour certains temps de décision. Toutefois, il n'exhibe pas un comportement stable en fonction du temps de décision. Le plus souvent, son pourcentage de réussite reste entre 70% et 90%. LRTP+I obtient 100% de réussite pour presque tous les temps de décision supérieurs à 130 ms. LRTP+J obtient 100% de réussite avec plus de 30 ms de temps de décision. Toutefois, tandis que le temps de décision augmente, une légère baisse des performances

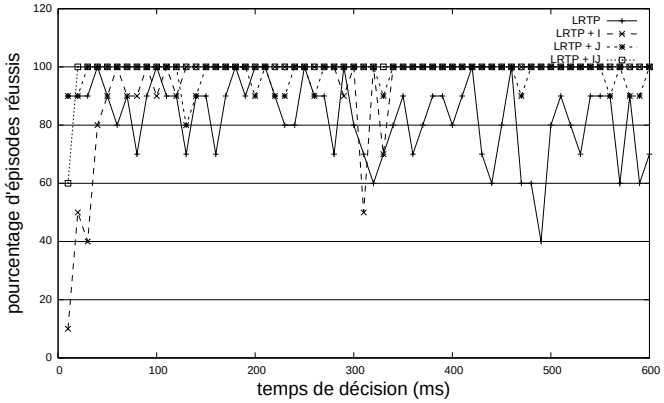


FIGURE 2.1 – Pourcentage de réussite en fonction du temps de décision (pb. 15 de Rovers, IPC5)

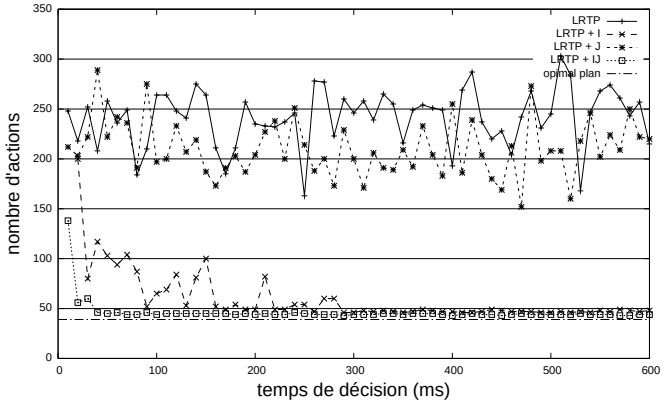


FIGURE 2.2 – Qualité du premier plan en fonction du temps de décision (pb. 15 Rovers IPC5)

se produit parfois. Les meilleures performances sont obtenues en utilisant les deux améliorations conjointement. Un pourcentage de réussite de 100% est obtenu pour un temps de décision de seulement 20 ms.

Qualité des plans solution

La figure 2.2 donne la longueur moyenne des plans solutions obtenus pour chacun des algorithmes. La longueur des plans trouvés par LRTP et LRTP+I est très médiocre, quel que soit le temps de décision (entre 150 et 300 actions). En effet, la longueur du plan optimal pour le problème Rovers 15 est de 46 actions. Par opposition, LRTP+J et LRTP+IJ produisent des plans très proches de la solution optimale même avec des temps de décision très petits de l'ordre de 200 ms pour LRTP+J et seulement 50 ms pour LRTP+IJ. L'amélioration (J) a donc un impact positif significatif sur la qualité des plans trouvés. L'amélioration (I) n'a d'impact positif que couplée avec l'amélioration (J).

2.4.3 Expérimentation 2. (Le domaine Rovers)

Ce paragraphe présente le pourcentage de réussite et la qualité moyenne des plans obtenus pour chaque problème du domaine Rovers pour les quatre variantes

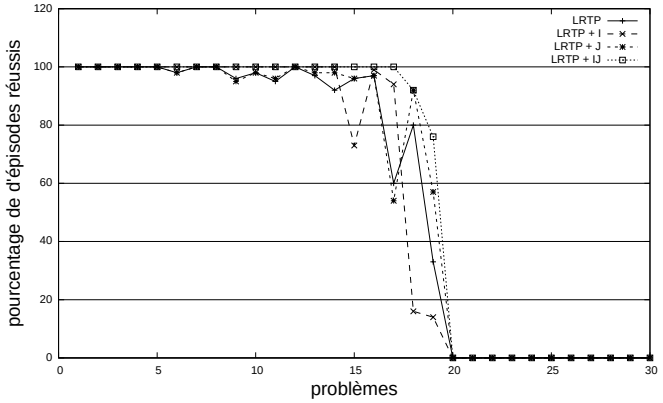


FIGURE 2.3 – Pourcentage de réussite pour le domaine Rovers en fonction du problème avec un temps de décision fixé à 100 ms

de LRTP. Le temps de décision est fixé ici à 100 ms pour bien visualiser l'écart de performances entre les différentes variantes de LRTP.

Pourcentage de réussite

La figure 2.3 montre le pourcentage de réussite de chaque variante de LRTP sur les 40 problèmes du domaine Rovers classés par ordre de difficulté lorsque le temps de décision est fixé à 100 ms. Une analyse globale montre que toutes les variantes de LRTP obtiennent 100% pour les cinq premiers problèmes. Pour les problèmes compris entre 5 et 15, on observe une légère baisse du taux de réussite pour les variantes LRTP et LRTP+I. Puis une forte baisse du pourcentage de réussite pour les problèmes de 15 à 20. Finalement à partir du problème 20, plus aucune variante ne trouve de plan de solution.

Maintenant si l'on regarde plus en détail chaque variante, on peut constater qu'avec l'amélioration (J), LRTP obtient des résultats très proches de ceux obtenus sans amélioration. Le pourcentage reste supérieur à 90% jusqu'au problème 16. Pour les problèmes 17 et 19, il atteint respectivement 54% et 57% de réussite. Enfin, à partir du problème 20, LRTP+J ne trouve pas de plan solution.

Avec l'amélioration (I), LRTP obtient 100% de réussite jusqu'au problème 14. Il reste supérieur à 90% jusqu'au problème 17, à l'exception du problème 15 dans lequel le pourcentage est de 79%. Pour les problèmes 18 et 19, le pourcentage est inférieur à 20%. Enfin, à partir du problème 20, aucun plan solution n'est trouvé.

Les meilleures performances sont obtenues encore une fois avec les deux améliorations (I) et (J). Nous pouvons voir que 100% des épisodes se terminent avec un plan solution jusqu'au problème 17. Ensuite, le pourcentage diminue à 92% dans le problème 18, et 76% dans le problème 19. Enfin, à partir du problème 20, aucun plan solution n'est trouvé.

Qualité des plans solution

La figure 2.4 présente la longueur moyenne des plans solutions obtenus par les quatre algorithmes dans tous les problèmes du domaine Rovers lorsque le temps de décision est fixé à 100 ms.

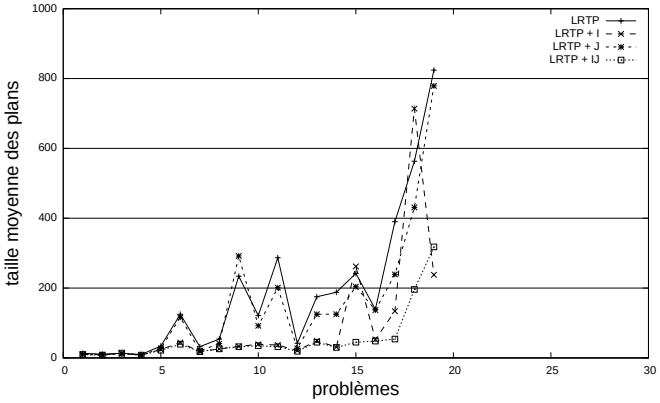


FIGURE 2.4 – Qualité moyenne des plans pour le domaine Rovers pour un temps de décision fixé à 100 ms

Jusqu'au problème 14, les meilleurs plans sont trouvés par LRTP+I et LRTP+IJ. Sur les mêmes problèmes, LRTP et LRTP+J obtiennent des plans solutions de longueur comparable, mais de qualité bien inférieure avec de grandes variations. Pour les problèmes de 15 à 20, LRTP+IJ se dégage nettement des autres variantes en produisant les meilleurs plans. Puis viennent LRTP+I, LRTP+J et finalement LRTP.

2.4.4 Expérimentation 3. (Résultats sur différents domaines)

Cette section présente les résultats obtenus par les quatre algorithmes sur tous les problèmes de 15 domaines issus de IPC-2, IPC-3, IPC-4 et IPC-5. Le temps de décision utilisé dépend du domaine considéré. Il a été défini suffisamment élevé pour permettre aux différentes variantes de LRTP de résoudre une majorité des problèmes de chaque domaine. Ainsi, il a été fixé à 100 ms dans le domaine DriverLog, 200 ms dans Pathways, 500 ms dans les domaines Blocksworld, Depots, Elevator, Freecell, Logistics, Pipesworld-no-Tankage, Rovers, satellite et Zenotravel, et finalement à 1000 ms dans Airport, Opentrack, PSR et TPP.

Pourcentage de réussite

La figure 2.5 présente le pourcentage de réussite moyen des algorithmes pour chaque domaine étudié. Nous pouvons observer que, dans la majorité des domaines, les meilleures performances sont obtenues avec les deux améliorations conjointement utilisées. Certaines exceptions sont observées dans Depots, Pipesworld et Blocksworld lorsque l'amélioration (J) est utilisée seule, celle-ci donnant des performances légèrement meilleures que lorsque les deux améliorations sont utilisées. Il peut également être observé qu'utiliser LRTP+J améliore presque toujours les performances, avec ou sans l'amélioration (I). Une exception est observée dans Satellite. Dans ce cas précis, utiliser une des améliorations diminue les performances alors qu'utiliser les deux les améliore.

Qualité des plans solution

La figure 2.6 donne la longueur moyenne des plans solutions trouvés en fonction du domaine. Nous pouvons voir que LRTP+IJ présente le meilleur comportement. Il est possible d'escompter un gain de 10% à 50% en utilisant la variante de LRTP+IJ

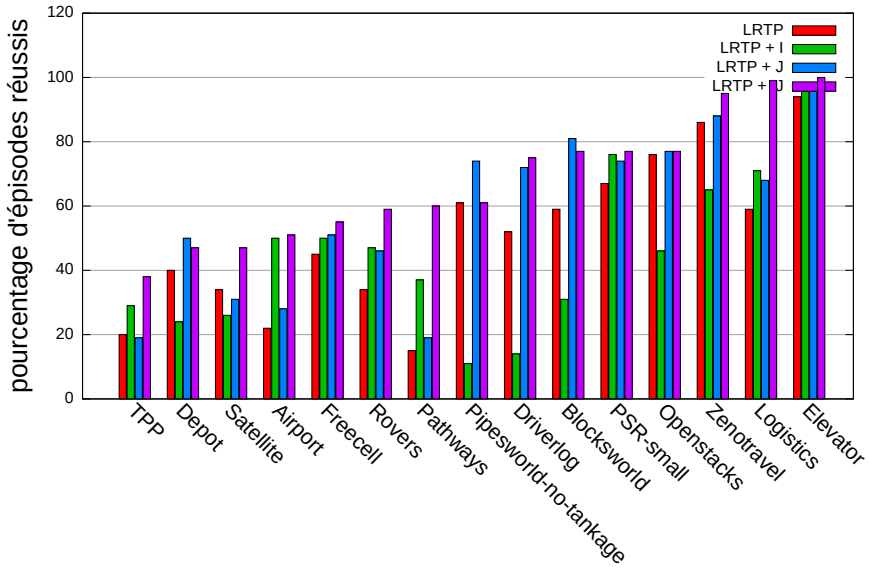


FIGURE 2.5 – Pourcentage de réussite sur plusieurs domaines IPC

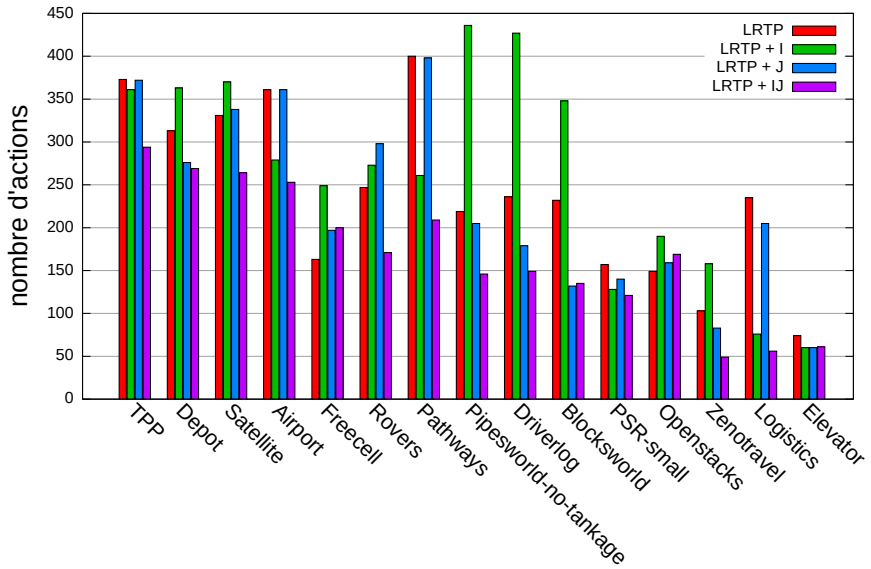


FIGURE 2.6 – Qualité moyenne des plans sur plusieurs domaines IPC

en fonction des domaines. De son côté, LRTP+I obtient des résultats médiocres sur Pipesworld, DriverLog et Blocksworld, domaines où l'agenda de buts est peu informatif. *A contrario*, la variante LRTP+J ne dégrade pas la qualité des plans produits, mais ne l'améliore pas non plus de manière significative.

2.4.5 Discussion

Nos expérimentations montrent que LRTP+I n'apparaît pas clairement comme une amélioration en termes de pourcentage de succès. On constate que les domaines pour lesquels LRTP+I apporte un gain sont ceux où l'agenda de buts est informatif. Dans le cas contraire, la variante de LRTP+I a plutôt tendance à détériorer les performances et pour certains domaines la qualité des plans solutions produits.

Par opposition, LRTP+J est clairement une amélioration de LRTP en termes de pourcentage de succès. Le fait de s'engager à exécuter plusieurs actions au cours d'un même temps de décision a pour effet d'accorder plus de temps pour les prochaines décisions et par conséquent leur permettre d'être de meilleure qualité. En revanche, LRTP+J a tendance à détériorer la qualité des plans solutions trouvés.

Finalement, LRTP+IJ est clairement une amélioration de LRTP+J, de LRTP+I et de LRTP. De plus, LRTP+IJ produit des plans solutions plus courts que les plans produits par LRTP+J et à un pourcentage de réussite plus élevé que les autres variantes. Nous pouvons en conclure que (I) et (J) doivent être utilisés simultanément.

2.5 Conclusion

Dans ce chapitre, nous avons présenté un nouvel algorithme de planification entrelaçant planification et exécution fondé sur les algorithmes de recherche temps réel appelé LRTP. Nous avons également présenté deux techniques pour améliorer la phase de planification temps réel : la première amélioration (I) s'appuie sur un agenda de buts pour classer les buts par ordre de difficultés croissantes et tente ensuite de les résoudre de manière incrémentale ; la seconde amélioration (J) consiste à choisir à chaque pas de décision une séquence d'actions qui sera exécutée et non plus une seule. Utilisées conjointement, ces deux techniques améliorent LRTP. Le pourcentage de succès est meilleur, les plans solutions plus courts, et par conséquent le temps nécessaire à leur exécution. En outre, ces deux techniques combinées permettent à LRTP de travailler avec des temps de décisions très courts, de l'ordre de quelques centaines de millisecondes.

Plusieurs pistes intéressantes de recherche sont ouvertes par ce travail. Dans le cadre des algorithmes de recherche temps réel, la fonction heuristique est mise à jour au fur et à mesure des épisodes. Nous pourrions envisager de manière identique de mettre à jour l'agenda de buts au fur et à mesure des épisodes à partir des plans solutions trouvés. Cela pourrait avoir un impact significatif en termes de performance notamment sur les domaines où il est difficile de calculer un agenda de buts informatif à partir de la seule description du problème de planification à résoudre. Finalement, il serait aussi intéressant de proposer et de comparer d'autres mécanismes de sélection d'actions s'appuyant par exemple sur la technique des *landmarks* proposée par (RICHTER, HELMET et WESTPHAL, 2008), ou encore sur des techniques de recherche de type *Monte-Carlo Tree Search* (KOCSIS et SZEPESVARI, 2006) développées spécifiquement pour prendre des décisions dans l'incertain. C'est cette dernière piste que nous proposons de suivre dans le chapitre suivant.

Chapitre 3

Planifier en temps réel en s'inspirant des algorithmes de recherche arborescente de Monte-Carlo

NOUS proposons dans ce chapitre une seconde contribution pour résoudre le problème de la planification en temps réel (PELLIER, BOUZY et MÉTIVIER, 2010a,b). Le point de départ de ce travail est d'appliquer UCT (*Upper Confidence for Tree*) (KOCSIS et SZEPESVARI, 2006), un algorithme efficace bien connu par les communautés de l'apprentissage automatique et des jeux, sur des problèmes de planification temps réel tels que présentés au chapitre 2. UCT est fondé sur une prise de décision de type bandit (AUER, CESA-BIANCHI et FISHER, 2002) couplée à une recherche arborescente de type Monte-Carlo. Dans le contexte de la planification temps réel, l'aspect intéressant d'UCT est sa propriété *anytime* au sens strict. À tout moment, UCT est en mesure de retourner la première action d'un plan, un plan partiel, un plan de solution, ou un plan optimal en fonction du temps qui lui est accordé. UCT a obtenu de très bons résultats dans le domaine des jeux, et plus particulièrement au jeu de Go avec le programme Mogo (GELLY et al., 2006). Dans ce contexte, UCT est très efficace malgré le coefficient de branchement élevé de l'arbre de recherche associé. Quelques travaux ont déjà regroupé planification d'actions et méthodes de Monte-Carlo. En 2006 par exemple, Chaslot et al. (CHASLOT et al., 2006) ont utilisé un algorithme proche d'UCT pour résoudre des problèmes de gestion de production. En 2009, Bjarnason et al. (BJARNASON, FERN et TADEPALLI, 2009) ont présenté une adaptation efficace d'UCT pour résoudre des problèmes probabilistes de planification. Finalement, Nakhost et Müller (NAKHOST et MÜLLER, 2009) ont proposé le planificateur ARVAND dans le cadre de la planification déterministe. ARVAND utilise une recherche arborescente fondée sur des simulations aléatoires de Monte-Carlo.

Dans la suite de ce chapitre, nous présentons tout d'abord rapidement l'algorithme UCT. Puis, nous présentons notre contribution : un nouvel algorithme de recherche heuristique fondé sur le calcul de moyennes pour la planification temps réel, appelé MHSP (*Mean-Based Heuristic Search Planning*). Cet algorithme associe les principes d'UCT et d'une recherche heuristique pour obtenir un planificateur temps réel. Nous terminons en proposant une évaluation de MHSP. Notre contribution est évaluée sur différents problèmes de planification et comparée aux algorithmes de recherche temps réel existants. Nos résultats mettent en évidence la capacité de MHSP à retourner en temps réel des plans qui tendent vers un plan optimal au cours du temps. Ils montrent de plus que MHSP est plus rapide et retourne des plans de

meilleure qualité que les algorithmes de la littérature.

3.1 Principe de la recherche arborescente avec UCT

UCT a obtenu de bons résultats dans les programmes jouant au jeu de Go et a été utilisé sous de nombreuses versions (CHASLOT et al., 2008). Tant qu'il reste du temps, UCT développe itérativement un arbre dans la mémoire de l'ordinateur en suivant les étapes suivantes :

1. À partir de la racine, parcourir l'arbre jusqu'à atteindre une feuille en utilisant la règle de sélection UCB (*Upper Confidence Bound*) (cf. équation 3.1) ;
2. Choisir une action et développer le nœud fils de la feuille choisie ;
3. Exécuter une simulation aléatoire à partir du nœud fils construit jusqu'à la fin du jeu et obtenir le retour, i.e., le résultat du jeu ;
4. Mettre à jour la valeur moyenne des nœuds parcourus avec le retour obtenu.

Avec un temps infini, la valeur du nœud racine converge vers la valeur min-max de l'arbre de jeu (KOCSIS et SZEPESVARI, 2006). La règle de sélection UCB donnée ci-dessous répond à la nécessité d'être optimiste lorsqu'une décision doit être prise dans l'incertain (AUER, CESA-BIANCHI et FISHER, 2002) :

$$N_{select} = \arg\max_{n \in N} \{m + C\sqrt{\frac{\log p}{s}}\} \tag{3.1}$$

où N_{select} est le nœud sélectionné, N est l'ensemble des nœuds fils, m est la valeur moyenne du nœud n, s est le nombre d'itérations passées par le nœud n, p est le nombre d'itérations passées par le nœud parent de n, et C est une valeur constante définie expérimentalement. L'équation (3.1) est la somme de deux termes : la valeur moyenne m et la valeur de biais UCB qui garantit l'exploration. Le respect de la règle de sélection UCB garantit l'exhaustivité et l'exactitude de l'algorithme.

3.2 L'algorithme MHSP

Notre contribution, l'algorithme MHSP, est une adaptation de l'algorithme UCT à la planification STRIPS (FIKES et NILSSON, 1971) comme définie section 1.4. Les grandes étapes de l'algorithme sont décrites à la figure 3.1. L'algorithme complet est donné Algo. 4. MSHP prend en paramètres : A l'ensemble des actions du problème à résoudre, s_0 l'état initial, g le but à atteindre et h la fonction heuristique à utiliser pour guider la recherche. $C[s]$ représente l'ensemble des fils de l'état s, $R[s]$ le retour cumulé de l'état s, $V[s]$ le nombre fois où l'état s a été visité et $P[s]$ le parent de s. Nous détaillons ci-dessous les modifications apportées à UCT.

3.2.1 Des retours obtenus par l'appel à une fonction heuristique

Dans le cadre du jeu de Go, les simulations aléatoires apportent des informations sur la structure de l'espace de recherche parce qu'elles se terminent toujours après un nombre limité de coups, et les valeurs retournées (gagné ou perdu) sont à peu près également réparties sur la plupart des positions du jeu. De plus, les valeurs retournées par les simulations correspondent aux valeurs de jeux terminés. En planification, il y a très peu d'états solutions. Par conséquent, les valeurs retournées par les simulations sont très peu informatives. La solution retenue pour pallier ce

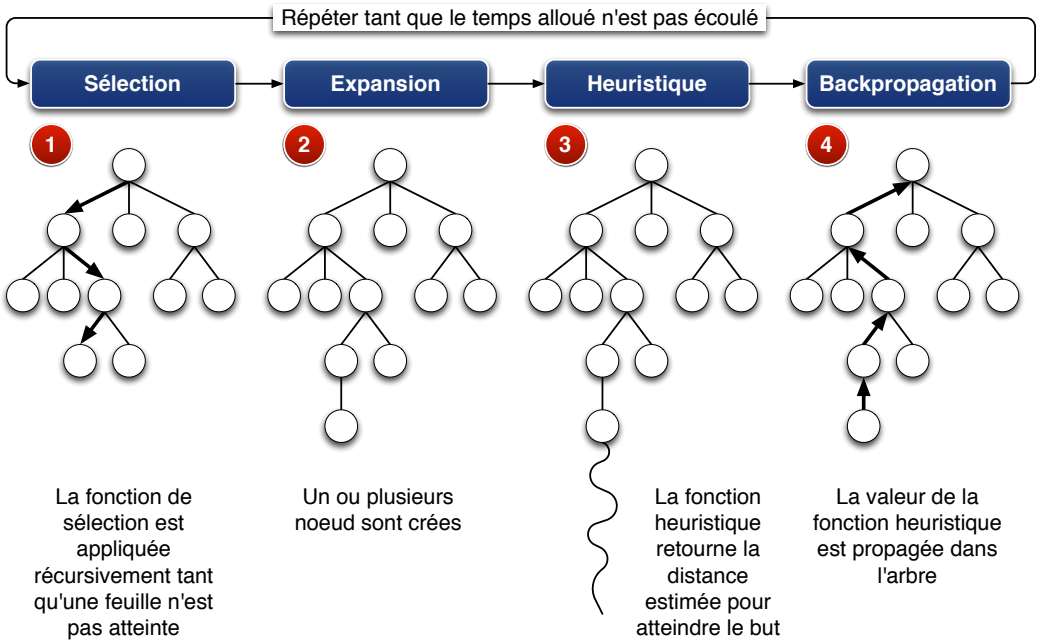


FIGURE 3.1 – Principe de l'algorithme MHSP

problème consiste à retourner, dans le cas où une simulation s'arrêterait sans avoir trouvé un état solution, la distance estimée par une fonction heuristique de l'état terminal de la simulation au but. Cette idée est ici reprise du planificateur ARVAND (NAKHOST et MÜLLER, 2009).

3.2.2 La suppression du biais d'exploration

La pratique des programmes de jeux montre que le biais UCB de l'équation (3.1) peut être supprimé, à condition que les valeurs moyennes des nœuds soient initialisées avec des valeurs suffisamment optimistes. Cette simplification supprime le problème de la détermination de la valeur du paramètre C. Une solution naïve consiste à initialiser les nœuds avec pour valeur initiale 0. Toutefois, cette valeur est certes optimiste, mais peu informative. Une solution plus pertinente consiste à initialiser les nœuds avec la valeur d'une fonction heuristique admissible. Cette valeur reste optimiste puisqu'elle ne surestime jamais la distance au but. Par ailleurs, cette valeur est beaucoup plus informative. De plus, le choix d'une heuristique admissible garantit la convergence vers le plan solution optimal. C'est donc cette dernière solution que nous avons retenue pour MSHP.

Dans MHSP, les retours sont négatifs ou nuls. Ils représentent l'opposé de la distance de l'état courant au but. Avec cette méthode d'initialisation, le meilleur nœud selon l'heuristique sera exploré en premier. Sa valeur sera réduite après quelques itérations quelle que soit son efficacité, et les autres nœuds seront alors examinés dans l'ordre donné par la fonction heuristique.

3.2.3 L'extraction des plans partiels et des plans solutions

Si le temps imparti est suffisant pour construire un état but, l'extraction d'un plan solution est réalisée simplement en sélectionnant depuis la racine les nœuds qui possèdent la meilleure valeur moyenne jusqu'à ce qu'une feuille soit atteinte. C'est

Algorithme 4 : MHSP(A, s_0, g, h)

```
1  C[s_0] ← ∅ ; R[s_0] ← h(s_0) ; V[s_0] ← 1 ; π ← nil
2  while has_time do
3      s ← s_0
4      ;; Step 1
5      while g ⊈ s and V[s] ≠ 1 do
6          s ← argmax_{s'∈C[s]}(R[s']/V[s'])
7      ;; Step 2 et 3
8      reward ← (R[s_0]/V[s_0]) + 1
9      if g ⊆ s then reward ← 0
10     else if V[s] = 1 then
11         foreach a ∈ A do
12             s' ← (s ∪ effects+(a)) − effects−(a)
13             C[s] ← C[s] ∪ {s'}
14             R[s'] ← h(s')
15             P[s'] ← s
16             V[s'] ← 1
17         if C[s] ≠ ∅ then
18             s ← argmax_{s'∈C[s]}(R[s'])
19             reward ← R[s]
20     ;; Step 4
21     l ← s
22     i ← 0
23     while s ≠ s_0 do
24         s ← P[s]
25         R[s] ← R[s] + (reward − i)
26         V[s] ← V[s] + 1
27         i ← i + 1
28     if g ⊆ l then
29         π' ← reconstruct_solution_plan()
30         if length(π) > length(π') then π ← π'
31 if π = nil then return reconstruct_best_plan()
32 else return π
```

la même procédure qu'utilise UCT pour extraire un plan solution de l'arbre de recherche (fonction *reconstruct_solution_plan()* ligne 29 Algo. 4). Dans le cas contraire, il s'agit d'extraire un plan partiel à partir de l'arbre de recherche qui a le plus de chance de nous conduire vers un état but. Nous avons fait le choix ici d'extraire non pas la séquence d'actions ayant la meilleure moyenne, mais la séquence d'actions qui a été la plus visitée (fonction *reconstruct_best_plan()* ligne 31 Algo 4). En effet, en s'appuyant sur les valeurs moyennes des nœuds, des nœuds nouvellement créés pourraient être choisis. La valeur moyenne n'est donc pas la valeur la plus pertinente pour guider l'extraction d'un plan partiel de l'arbre de recherche.

3.2.4 La gestion des puits de l'espace de recherche

Enfin, nous devons discuter de la façon dont MHSP gère les puits de l'espace de recherche. Normalement, la valeur heuristique calculée pour les états puits est infinie. La valeur infinie doit alors servir à mettre à jour la valeur moyenne de tous les nœuds du nœud puits à la racine de l'arbre de recherche. Ce cas est critique puisque la propagation de cette valeur conduirait à invalider toute une branche de l'arbre de recherche alors que nous voulons juste empêcher MHSP de passer dans les nœuds puits. Pour résoudre ce problème, MSHP rétro propage une valeur calculée à partir de la valeur du nœud racine et un coefficient ajustable afin de sanctionner le nœud puits, ainsi que les nœuds parcourus pour l'atteindre depuis la racine.

3.3 Expérimentation et évaluation

Le but de ces expériences est de comparer la stratégie de sélection d'actions utilisées par MHSP avec les stratégies de sélection d'actions classiquement utilisée par les algorithmes de recherche temps réel présentés au chapitre 2.

3.3.1 Stratégies et domaines de planification retenus

La stratégie de sélection d'actions de MHSP est comparée à deux autres stratégies : A* et BFS (*Best First Search*). Nous avons choisi A*, car celui-ci est l'algorithme de référence en planification. C'est la stratégie par défaut utilisée par l'algorithme LRTP présenté au chapitre 2. La stratégie est ici de type *meilleur d'abord*. Elle vise à minimiser la fonction heuristique classique f de A*. A* étend les nœuds de la liste restant à explorer dans l'ordre donné par f. De ce fait, A* peut être interrompu à tout moment. Lorsque A* est arrêté, le nœud ayant la meilleure valeur heuristique dans la liste des nœuds restant à explorer est choisi. Le chemin de la racine de l'arbre de recherche à ce nœud est retourné comme la meilleure séquence d'actions à exécuter dans le temps imparti. BFS est l'algorithme classique utilisé pour la sélection d'actions par les algorithmes de recherche temps réel. Lorsque BFS est arrêtée la séquence d'actions de la racine au nœud feuille ayant la meilleure valeur heuristique est retournée.

Les domaines de planification retenus pour l'évaluation sont : Ferry, Gripper et Satellite. Ces domaines sont suffisamment simples pour effectuer les expérimentations relativement rapidement. Pour chacun de ces domaines classiques, nous avons sélectionné 20 problèmes classés par ordre de difficulté.

3.3.2 Protocole expérimental

Trois tests ont été mis en œuvre afin de souligner les bonnes propriétés de MHSP par rapport aux autres stratégies. Ces tests ont été effectués sur un ordinateur équipé d'un processeur Intel Core 2 Quad 6600 (2,4 GHz) avec 4 Go de mémoire vive. Les implémentations de MHSP et des autres stratégies utilisées pour ces expériences sont écrites en Java et s'appuient sur la bibliothèque PDDL4J [1]. Nous avons utilisé pour les expérimentations l'heuristique de FastForward (HOFFMANN et NEBEL, 2001). Une étude des performances de MSHP en fonction de l'heuristique choisie est disponible dans (PELLIER, BOUZY et MÉTIVIER, 2010b).

1. https://github.com/pellierd/pddl4j

Test 1. (Évaluation sans apprentissage)

Le test 1 donne pour les problèmes des 3 domaines Ferry, Gripper et Satellite retenus et pour des temps de décision croissants : le temps moyen par épisode pour trouver un plan solution, la longueur moyenne, maximale et minimale des plans solutions trouvés comparativement au plan optimal ainsi que le nombre d'épisodes qui se terminent en ayant trouvé un plan solution. Pour ce test, 50 épisodes ont été réalisés. Le temps alloué par épisode est de 300 secondes. Aucun apprentissage n'est réalisé entre les épisodes.

Test 2. (Évaluation avec apprentissage)

Le test 2 réexécute le test 1 avec apprentissage en appliquant la règle de mise à jour utilisée par LRTP (cf. équation 2.1) sur les nœuds s visités au cours des épisodes. Le test 2 entend montrer l'efficacité des trois sélecteurs d'actions sur la convergence des plans solutions vers des plans optimaux lorsque le nombre d'épisodes augmente.

Test 3. (Évaluation de la qualité des décisions)

Le test 3 propose une évaluation de la qualité des décisions prises en fonction du temps alloué par décision en évaluant la qualité du plan partiel construit à chaque décision. La qualité des plans partiels est estimée grâce à deux distances : la distance à l'objectif et la distance à l'optimum.

1. *La distance à l'objectif* d'un plan partiel est la longueur du plan optimal reliant l'état final de ce plan partiel à l'état but. Lorsque la distance à l'objectif diminue, le plan partiel a été construit dans la direction appropriée. Lorsque la distance de l'objectif est nulle, le plan partiel est un plan solution.
2. *La distance à l'optimum* d'un plan partiel est la longueur du plan partiel, plus la distance à l'objectif du plan partiel, moins la longueur du plan optimal. Quand la distance à l'optimum d'un plan partiel est nulle, le plan partiel est le début d'un plan optimal. La distance à l'optimum d'un plan solution est la différence entre sa longueur et la longueur optimale. La distance à l'optimum du plan vide est zéro. La distance à l'objectif et la distance à l'optimum d'un plan optimal sont nulles. Réciproquement, si la distance à l'objectif et la distance à l'optimum d'un plan partiel sont nulles, alors ce plan partiel est un plan optimal. Pour chaque problème, les résultats sont présentés avec des graphiques montrant la distance à l'objectif et la distance à l'optimum du plan partiel en fonction du temps d'exécution. Ces distances sont calculées en appelant un planificateur optimal (A*).

3.3.3 Résultats expérimentaux

Test 1. (Évaluation sans apprentissage)

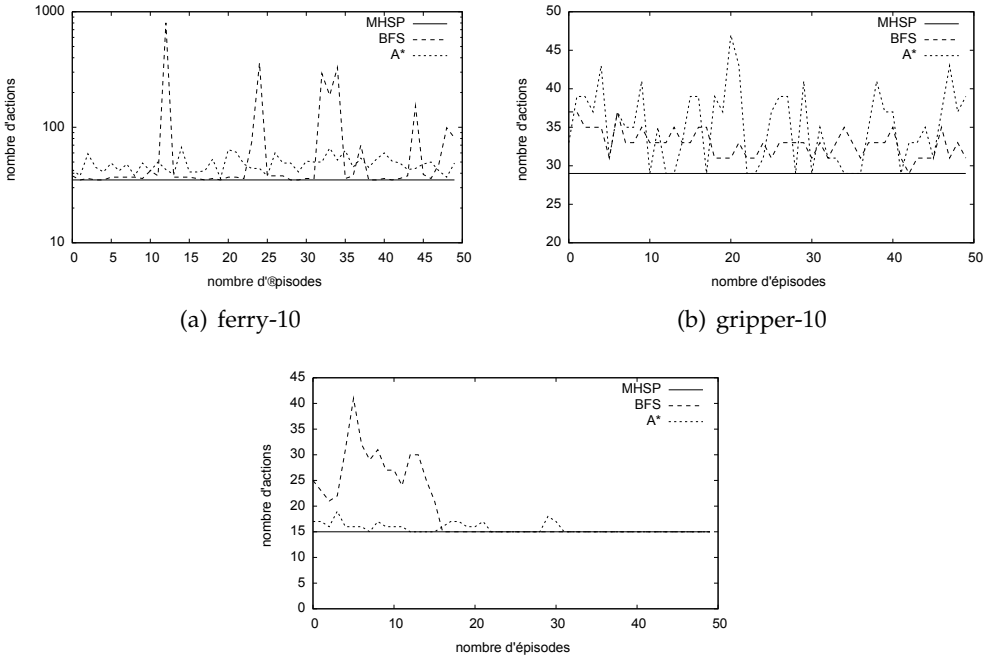
Le tableau 3.1 présente les résultats du test 1 sur les domaines retenus. Sur le domaine Ferry quelque soit le problème testé et le temps de décision MHSP surpasse en temps les autres stratégies A* et BFS dans cet ordre. On retrouve les mêmes résultats si l'on regarde la longueur des plans solutions trouvés. MHSP trouve même avec un temps de décision très faible (40 ms) le plan optimal à chaque épisode pour les problèmes simples. Ce n'est pas le cas de A* et BFS qui trouvent des plans solutions relativement loin de l'optimal même s'ils s'en rapprochent lorsque le temps

Problème	**algo.**	**décision**	**temps moy.**	**long. moy.**	**long. opt.**	**long. max.**	**long. min.**	**% échecs**
ferry-05	A*	40	1,09	26,26	18	277	19	0
ferry-05	BFS	40	8,91	27,76	18	567	18	16
ferry-05	MHSP	40	0,59	18,02	18	19	18	0
ferry-10	A*	200	97,9	184,94	35	807	42	32
ferry-10	BFS	200	9,05	41,35	35	109	35	0
ferry-10	MHSP	200	6,26	35,46	35	36	35	0
ferry-15	A*	2000	157,85	31,95	51	88	58	55
ferry-15	BFS	2000	103,75	51,45	51	52	51	0
ferry-15	MHSP	2000	86,45	52,45	51	53	51	0
ferry-20	A*	4000	–	–	73	–	–	100
ferry-20	BFS	4000	–	–	73	–	–	100
ferry-20	MHSP	4000	260,49	74,87	73	78	73	0
gripper-05	A*	50	0,56	15,04	15	17	15	0
gripper-05	BFS	50	0,71	15	15	15	15	0
gripper-05	MHSP	50	0,49	15	15	15	15	0
gripper-10	A*	165	92,28	140,04	29	651	33	36
gripper-10	BFS	165	7,86	37	29	37	37	0
gripper-10	MHSP	165	5,08	29	29	29	29	0
gripper-15	A*	450	160,1	47,54	45	229	77	70
gripper-15	BFS	450	31,42	46,52	45	47	45	0
gripper-15	MHSP	450	38,53	54,88	45	55	53	0
gripper-20	A*	1100	–	–	59	–	–	100
gripper-20	BFS	1100	102,76	61	59	61	61	0
gripper-20	MHSP	1100	134,86	73,92	59	75	73	0
satellite-05	A*	300	3,49	15,08	15	18	15	0
satellite-05	BFS	300	20,28	63,72	15	522	19	0
satellite-05	MHSP	300	3,01	15	15	15	15	0
satellite-10	A*	2000	–	–	29	–	–	100
satellite-10	BFS	2000	–	–	29	–	–	100
satellite-10	MHSP	2000	67,912	31,0	29	31	31	0

TABLE 3.1 – Test 1 – Comparaison des différentes stratégies de sélection d'actions MSHP, A* et BFS en ce qui concerne le temps moyen des épisodes, de la longueur moyenne, maximale et minimale des plans solutions et du pourcentage d'échecs, sans apprentissage

de décision augmente. Si l'on regarde maintenant le taux d'échec, on constate que 100% des épisodes effectués avec MSHP trouvent un plan solution, ce qui n'est pas le cas pour A* et BFS. À partir du problème 20, plus aucun épisode utilisant ces deux stratégies ne se termine en trouvant un plan solution.

Les mêmes résultats se retrouvent sur les domaines Gripper et Satellite. MSHP surpasse en temps et en qualité des plans solutions les autres stratégies. Pour résumer le test 1, MHSP trouve des plans plus courts que A* ou BFS, et MHSP est plus rapide que A* et BFS.



(a) ferry-10

(b) gripper-10

(c) satellite-05

FIGURE 3.2 – Test 2 – Convergence de MSHP, A* et BFS en fonction du nombre d'épisodes

Test 2. (Évaluation avec apprentissage)

Le tableau 3.2 montre les performances des trois stratégies de sélection lorsqu'un apprentissage est réalisé entre les épisodes. Par rapport aux résultats de la table 3.1, nous pouvons observer que l'apprentissage améliore le plus souvent la qualité des meilleurs plans. En effet, à l'exception de BFS dans gripper-20, la longueur minimale des plans solutions est toujours plus petite avec apprentissage que sans, si elle n'est pas déjà optimale dans le test 1.

De plus, l'apprentissage a permis à BFS et A* de trouver des plans solutions dans ferry-15. Aucun d'entre eux n'est un plan optimal, mais le meilleur plan de BFS est à deux actions de l'optimal. L'apprentissage a également permis à MHSP de devenir optimal dans satellite-05. L'apprentissage améliore donc la qualité des plans solutions des trois stratégies.

Afin d'illustrer le comportement des trois stratégies de sélection d'actions en fonction du nombre d'épisodes, la figure 3.2 montre l'évolution de la longueur des plans en fonction du nombre d' épisodes sur trois problèmes : ferry-10, gripper-10 et satellites-5. Sur ces problèmes, MHSP atteint un plan optimal dès le premier épisode et n'évolue plus. Par opposition, A* et BFS ne trouvent pas les plans optimaux pour les problèmes ferry-10 et gripper-10 même après plus de 50 épisodes, et sont très instables.

Contrairement à ce que l'on aurait pu croire, l'apprentissage n'améliore pas de manière significative la longueur des plans lorsque le nombre d'épisodes augmente. Ceci s'explique par le fait que la fonction heuristique utilisée est relativement performante pour les problèmes considérés. Il y a donc très peu de mise à jour de la fonction heuristique entre les épisodes.

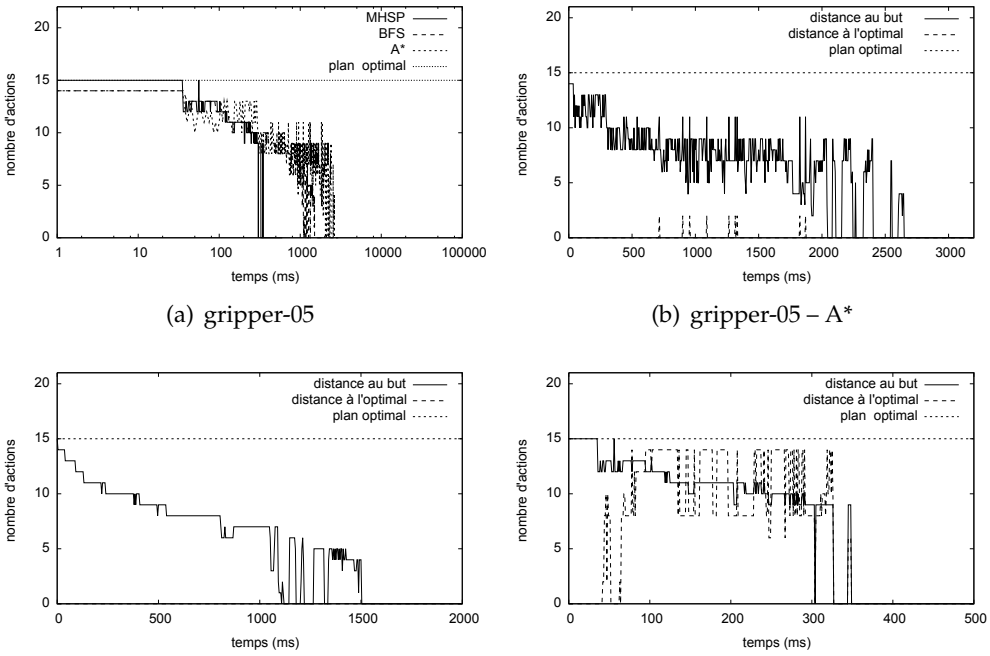
problème	algo,	décision	temps moy.	long. moy.	long. opt.	long. max.	long. min.	% échecs
ferry-05	A*	40	0,68	19,22	18	31	18	0
ferry-05	BFS	40	12,03	133,18	18	532	18	14
ferry-05	MHSP	40	0,48	18,02	18	19	18	0
ferry-10	A*	200	9,36	48,12	35	67	37	0
ferry-10	BFS	200	17,22	78,5	35	892	35	0
ferry-10	MHSP	200	6,24	35	35	35	35	0
ferry-15	A*	2000	112,03	62,36	51	81	57	55
ferry-15	BFS	2000	104,31	51,75	51	53	51	0
ferry-15	MHSP	2000	87,39	52,8	51	53	51	0
ferry-20	A*	4000	247,32	121,1	73	146	107	0
ferry-20	BFS	4000	284,44	73,8	73	75	75	35
ferry-20	MHSP	4000	255,31	73,6	73	73	73	15
gripper-05	A*	50	0,45	15	15	15	15	0
gripper-05	BFS	50	35,72	34,06	15	615	15	68
gripper-05	MHSP	50	0,48	15	15	15	15	0
gripper-10	A*	165	76,48	35,04	29	43	29	0
gripper-10	BFS	165	9,96	32,22	29	37	29	0
gripper-10	MHSP	165	4,8	29	29	29	29	0
gripper-15	A*	450	56,32	54,32	45	57	49	14
gripper-15	BFS	450	36,92	54,8	45	318	45	4
gripper-15	MHSP	450	36,52	46,76	45	55	45	0
gripper-20	A*	1100	–	–	59	–	–	100
gripper-20	BFS	1100	132,44	74,12	59	75	73	2
gripper-20	MHSP	1100	99,47	59,06	59	63	73	0
satellite-05	A*	300	3,54	15,62	15	18	15	0
satellite-05	BFS	300	6,1	18,98	15	41	19	0
satellite-05	MHSP	300	3,17	15	15	15	15	0
satellite-10	A*	2000	–	–	29	–	–	100
satellite-10	BFS	2000	–	–	29	–	–	100
satellite-10	MHSP	2000	65,612	31,2	29	32	30	0

TABLE 3.2 – Test 2 – Comparaison des différentes stratégies de sélection d'actions MSHP, A* et BFS en ce qui concerne le temps moyen des épisodes, la longueur moyenne, maximale et minimale des plans solutions et pourcentage d'échecs, avec apprentissage

Test 3. (Évaluation de la qualité des décisions)

Le test 3 évalue la qualité des plans partiels disponibles à la fin de la phase de sélection d'action en fonction du temps donné à la décision. La figure 3.3(a) donne un aperçu de la distance à l'objectif des trois algorithmes en fonction du temps de décision sur une échelle logarithmique. Les figures 3.3(b), 3.3(c) et 3.3(d) montrent la distance à l'objectif et la distance à l'optimum pour chaque algorithme en fonction du temps de décision sur le problème gripper-05.

Les résultats montrent que A* a besoin d'un temps de décision d'au moins 2650ms pour trouver systématiquement des plans optimaux, tandis que BFS a besoin d'au moins 1504ms et MHSP de seulement 349ms. Quel que soit le temps de décision, A* fournit des plans partiels proches de l'optimal. BFS ne fournit que



(a) gripper-05

(b) gripper-05 – A*

(c) gripper-05 – BFS

(d) gripper-05 – MHSP

FIGURE 3.3 – Test 3 – Comparaison de qualité des plans solutions en fonction de la stratégie de sélection MHSP, A* et BFS

algorithme	ferry-05	gripper-05	satellite-05
A*	808	2650	8180
BFS	-	1504	-
MHSP	288	349	1710

TABLE 3.3 – Temps (ms) de décision pour trouver le plan optimal dans les domaines Ferry, Gripper et Satellite en fonction de la stratégie de sélection d'actions : MHSP, A* et BFS

des plans partiels optimaux. Enfin, le tableau 3.3 résume ces résultats et ajoute les résultats sur ferry-05 et satellite-05. Comme dans Gripper, on peut voir que MHSP est le plus rapide des algorithmes pour trouver des plans optimaux.

3.4 Conclusion

Dans ce chapitre, nous avons présenté et étudié un nouvel algorithme de recherche heuristique fondé sur le calcul de valeurs moyennes pour la recherche temps réel, appelé MHSP, adapté dans le contexte de la planification classique. Cet algorithme combine une recherche heuristique et les principes de l'algorithme UCT, i.e., les valeurs des états reposant sur les retours moyens et l'optimisme face à l'incertain.

MHSP calcule des valeurs moyennes pour fonder ses décisions et non des valeurs heuristiques directes. Cela signifie que la valeur heuristique associée à un nœud dépend de tous les nœuds explorés au-delà de ce nœud et pas seulement du nœud. MHSP se focalise donc sur les actions permettant d'atteindre des nœuds globalement bons, et non sur l'action permettant d'atteindre le nœud ayant la meilleure

valeur heuristique. Dans un contexte temps réel, cette stratégie limite l'impact des valeurs heuristiques erronées. Elle permet d'explorer plus subtilement l'arbre de recherche. Plus le problème est complexe, plus cet effet est visible.

Trois tests ont été conçus afin de comparer MHSP, A * et BFS. Le premier a donné un aperçu de l'efficacité globale de chaque algorithme à trouver de bons plans pour différents problèmes issus des domaines Ferry, Gripper et Satellite. Nous avons montré que la stratégie de sélection d'actions de MHSP est globalement meilleur que celle de A* et de BFS. Le deuxième test a montré que l'apprentissage améliore la qualité des plans obtenus par MHSP. Ce test a montré par ailleurs que MHSP tend à converger très rapidement vers un plan optimal, tandis qu'A* et BFS restent sous-optimaux et instables. Enfin, le troisième test a permis d'évaluer la capacité de MSHP à effectuer en temps réel la sélection d'une action, ou d'un plan partiel pertinent. Les résultats ont mis en évidence que, le temps de décision augmentant, MHSP est beaucoup plus rapide à fournir des plans optimaux que les stratégies A* et BFS.

L'algorithme UCT a été développé initialement pour prendre des décisions dans des environnements incertains et non déterministes. Les résultats prometteurs présentés dans ce chapitre pour résoudre des problèmes déterministes de planification en temps réel ouvrent donc une voie naturelle d'extension de ce travail dans un cadre à la fois temps réel et non-déterministe encore inexploré dans la littérature.

Chapitre 4

Planifier lorsque le but change

DANS des environnements dynamiques, il n'est pas rare que des événements conduisent à une modification du but initialement confié à un système autonome. Ce cas se produit notamment lorsqu'un système autonome est en interaction avec un opérateur humain qui peut venir à tout instant modifier le but précédemment assigné. La littérature distingue principalement deux manières d'aborder ce problème (COX et VELOSO, 1998 ; FOX et al., 2006 ; KAMBHAMPATI et HENDLER, 1992 ; KROGT et WEERDT, 2005 ; NEBEL et KOEHLER, 1995) : planifier un nouveau plan en partant de zéro ou réparer le plan courant pour prendre en compte le nouveau contexte. Bien qu'en théorie ces deux approches soient équivalentes en ce qui concerne le coût dans le pire des cas (NEBEL et KOEHLER, 1995), les résultats expérimentaux montrent que l'approche qui consiste à réparer un plan existant est plus efficace que celle qui consiste à replanifier sans réutiliser les informations provenant des précédentes recherches (KROGT et WEERDT, 2005).

Notre contribution dans ce chapitre s'inscrit dans cette seconde approche. Nous proposons un nouvel algorithme de planification appelé MGP (*Moving Goal Planning*) (PELLIER, FIORINO et MÉTIVIER, 2013, 2014) pour la planification en boucle fermée entrelaçant exécution et planification où la recherche d'un plan est vue comme une recherche heuristique de type MTS (*Moving Target Search*) (ISHIDA et KORF, 1991). Les algorithmes de recherche heuristique de type MTS ont été très peu utilisés dans le cadre de la planification de tâches. Ceci s'explique en partie par le fait qu'il a longtemps été considéré comme difficile de trouver des fonctions heuristiques à la fois informatives et facilement calculables en planification. Ce n'est que vers la fin des années 90 que des planificateurs tels que HSP et HSP-r (BONET et GEFFNER, 1999, 2001) ont renversé la tendance en fournissant des méthodes génériques et efficaces pour calculer de telles fonctions heuristiques en temps polynomial. Ceci a eu pour conséquence de faire évoluer de manière quasiment indépendante ces deux domaines de recherche. Il nous apparaît dorénavant important de capitaliser sur les récentes avancées dans ces deux domaines pour proposer de nouveaux algorithmes de planification entrelaçant planification et exécution.

Le chapitre est organisé selon le plan suivant : la première partie présente succinctement l'état de l'art sur les algorithmes de type MTS, la partie 2 présente formellement le problème considéré et décrit notre contribution, l'algorithme MGP ; la partie 3 présente les résultats expérimentaux montrant l'efficacité de notre contribution ; finalement, la partie 4 présente nos conclusions sur le travail réalisé.

4.1 La recherche de cible mouvante

Les algorithmes de recherche de cible mouvante (MTS) ont été développés dans le cadre de la planification de trajectoires. Ils entrelacent recherche d'un chemin et

exécution : un agent *chasseur* poursuit une cible mouvante aussi appelée *proie* dans une grille à deux dimensions ou une carte. Depuis les travaux pionniers d'Ishida (ISHIDA, 1998 ; ISHIDA et KORF, 1991, 1995), les approches de type MTS peuvent se partager principalement en deux catégories selon la stratégie sur laquelle elles s'appuient pour réutiliser l'information des précédentes recherches.

La première stratégie consiste à utiliser une fonction heuristique pour guider la recherche et apprendre le chemin le plus court entre des couples de points sur une carte. À chaque nouvelle recherche, la fonction heuristique est plus informative et la recherche devient plus performante. Dans sa version originale, l'algorithme MTS est une adaptation de l'algorithme LRTA* (*Learning Real-Time A**) (KORF, 1990) que nous avons déjà évoqué dans les précédents chapitres. Cette approche est complète lorsque les déplacements de la cible sont plus lents que celui de l'agent. Toutefois, elle n'est pas sans inconvénient à cause des dépressions de la fonction heuristique et de la perte d'information induite par les déplacements de la cible (MELAX, 1993). Actuellement, l'état de l'art des algorithmes fondés sur cette première stratégie sont des variantes de AA* (KOENIG et LIKHACHEV, 2005), MTAA* (KOENIG, LIKHACHEV et SUN, 2007) et GAA* (SUN, KOENIG et YEOH, 2008). MTAA* adapte et apprend la fonction heuristique à partir des déplacements de la cible pour améliorer la convergence. GAA* généralise MTAA* avec la prise en compte d'actions auxquelles on a associé des coûts qui augmentent et diminuent au cours du temps. MTAA* et GAA* nécessitent tous deux l'utilisation de fonctions heuristiques admissibles pour garantir leur correction et leur complétude.

La seconde stratégie consiste à utiliser incrémentalement l'arbre de recherche entre deux recherches successives. Le premier algorithme fondé sur cette stratégie est D* (STENZ, 1995). Ses deux principaux successeurs sont décrits dans (KOENIG et LIKHACHEV, 2002 ; SUN, YEOH et KOENIG, 2010b). Ces algorithmes ont été conçus et développés pour calculer rapidement des trajectoires dans des environnements inconnus et dynamiques. Ils s'appuient tous sur une recherche en chaînage arrière. Ils obtiennent de bons résultats lorsque l'environnement évolue peu au cours du temps. En revanche, ils sont surclassés par l'approche naïve qui consiste à appeler successive l'algorithme A* (HART, NILSSON et RAPHAEL, 1968) à chaque fois que la cible se déplace (SUN, KOENIG et YEOH, 2008) lorsque les seules modifications portent sur le but. Récemment, un autre algorithme s'appuyant sur une recherche en chaînage avant appelé FRA* a été proposé (SUN, YEOH et KOENIG, 2009). Cet algorithme n'est pas capable de prendre en compte les changements de l'environnement, mais obtient de très bons résultats quand il s'agit de prendre en compte les déplacements d'une cible au cours du temps. Chaque fois que la cible se déplace, FRA* adapte rapidement l'arbre de recherche précédemment construit à la nouvelle position de la cible et rappelle la fonction de recherche A* sur le nouvel arbre de recherche. FRA* est actuellement l'algorithme MTS le plus efficace. Toutefois, l'adaptation de l'arbre est largement dépendante de la modélisation de l'environnement. FRA* considère en effet que l'environnement est modélisé sous la forme d'une grille, ce qui en limite les applications dans le cas général. Afin de pallier cet inconvénient, une variante de cet algorithme appelée GFRA* (SUN, YEOH et KOENIG, 2010a) a été proposée pour fonctionner dans des environnements modélisés par des graphes quelconques, y compris des treillis d'états utilisés pour la navigation de véhicules terrestres autonomes. En outre, GFRA* possède une autre caractéristique intéressante : il peut être utilisé avec une fonction heuristique non admissible, ce qui est souvent le cas en planification.

4.2 Problème, approche et implémentation

Dans cette partie, nous présentons tout d'abord formellement le problème traité. Dans un deuxième temps, nous présentons notre contribution, l'algorithme MGP, et détaillons sa stratégie de recherche. Finalement, nous introduisons deux stratégies qui permettent de retarder le déclenchement d'une nouvelle recherche lorsque le but évolue dynamiquement : *Open Check* et *Plan Follow*, et terminons en présentant la stratégie incrémentale de mise à jour de l'arbre de recherche.

4.2.1 Modélisation du problème

Pour définir notre problème, nous nous plaçons dans le cadre des hypothèses de la planification STRIPS telles que définies section 1.4. Conceptuellement, résoudre un problème de planification entrelaçant planification et exécution dans lequel le but évolue au cours du temps revient à résoudre une succession de problèmes de planification où l'état initial et le but changent.

Définition 4.1 (Problème de planification avec but mouvant)**.** *Un problème de planification en boucle fermée avec but mouvant à l'instant t est un tuple $P = (A, s_t, g_t, \delta)$ où A est l'ensemble des actions qui peuvent être exécutées, s_t et g_t sont respectivement l'état courant de l'agent et le but à atteindre et δ la fonction qui détermine la manière dont le but évolue en fonction du temps.*

Nous supposons dans la suite (1) qu'à chaque pas de temps t, une action est exécutée pour atteindre le but courant g_t faisant ainsi évoluer l'état courant s_t et (2) que l'agent qui planifie ne possède aucune information sur la fonction δ qui fait évoluer le but au cours du temps. Cette hypothèse est commune aux algorithmes MTS et à notre approche.

Définition 4.2 (Plan Solution)**.** *Un plan $\pi = \langle a_0, \ldots, a_n \rangle$ est solution d'un problème de planification avec but mouvant $P = (A, s_t, g_t)$ au pas de temps t ssi $g_t \subseteq s_t$ tel que $s_t = \gamma(s_0, \pi)$ où s_0 est l'état initial du problème à $t = 0$.*

Nous préciserons formellement dans le paragraphe 4.3 la fonction δ retenue pour réaliser l'évaluation empirique de notre approche.

4.2.2 Approche et principe de l'algorithme

Le pseudo-code de MGP est donné par l'algorithme 5. MGP prend en entrée un problème de planification (A, s_0, g_0). Les variables g et s représentent respectivement le but courant et l'état courant ayant pour valeur initiale g_0 et s_0. i est un compteur qui permet de comptabiliser le nombre de recherches effectuées.

MGP appelle itérativement une procédure de recherche (ligne 3) tant que le but courant n'a pas été atteint. Cette procédure construit un arbre de recherche en chaînage avant dans un espace d'états. Si le but courant n'est pas atteignable, la recherche échoue impliquant l'échec de MGP également (ligne 3). Sinon, MGP extrait un plan à partir de l'arbre de recherche construit (lignes 4-5) et essaie de retarder autant que possible un nouvel appel à la procédure de recherche, i.e., une nouvelle expansion de l'arbre de recherche (boucle While et procédure CanDelayNewSearch ligne 4). Cette boucle prend fin lorsque le but évolue et n'est plus présent dans l'arbre de recherche précédemment construit. La procédure CanDelayNewSearch s'appuie sur deux critères pour déterminer si une recherche peut être retardée : (1) elle vérifie que le but n'est pas déjà atteignable, i.e., n'est pas présent dans l'espace de recherche

Algorithme 5 : MGP(A, s_0, g_0)

```
1  s ← s_0, g ← g_0, i ← 1
2  while g ⊈ s do
3      if Search(A, s, g, i) = FAILURE then return FAILURE
4      while CanDelayNewSearch(s, g) do
5          Extract plan π = ⟨a_1, ..., a_n⟩ from the search tree
6          while (g ⊈ s and g ⊆ γ(s, π)) or (PlanFollow(s, g) and π ≠ ∅) do
7              a ← extract the first action of π
8              Executes a
9              s ← γ(s, a)
10             π ← extract the trail of π (π without its first action)
11             g ← UpdateGoal(g)
12         if g ⊆ s then return SUCCESS
13         DeleteStatesOutOfTree(s)
14     UpdateSearchTree(s, g)
15     i ← i + 1
```

(procédure OpenCheck) ; (2) elle estime l'éloignement du nouveau but par rapport à l'ancien et renvoie vrai si cette distance est inférieure à une borne donnée (procédure PlanFollow). Tant que le but est accessible à partir du plan précédemment extrait ou que le but n'a pas évolué de manière significative, MGP exécute alors les actions de ce plan (lignes 6). Dans le cas où MGP atteint le but courant, MGP se termine avec succès (ligne 12). Dans le cas contraire, le but a évolué (ligne 11) et MGP doit élaguer son arbre de recherche. La fonction UpdateGoal simule l'évolution du but et retourne le nouveau but s'il a évolué. L'arbre de recherche est alors le sous-arbre dont la racine est représentée par son état courant s (DeleteStatesOutOfTree, ligne 13). Si le nouveau but est présent dans ce sous-arbre, il n'est pas nécessaire de lancer une nouvelle recherche. MGP extrait directement un nouveau plan à partir de celui-ci et exécute ses actions pour atteindre le nouveau but. Sinon, MGP met à jour les valeurs heuristiques des nœuds de l'arbre de recherche en estimant la distance pour atteindre le nouveau but (ligne 14) et finalement étend l'arbre de recherche ainsi modifié (ligne 3).

4.2.3 Implémentation de l'algorithme

Dans cette partie, nous présentons les détails relatifs à l'implémentation de MGP. L'arbre de recherche est représenté par deux listes notées OPEN et CLOSED : la liste OPEN contient les états feuilles de l'arbre de recherche restant à explorer et la liste CLOSED, la liste des états déjà explorés. Dans un premier temps, nous présentons la stratégie de recherche de MGP. Puis, nous présentons la procédure CanDelayNewSearch utilisée pour retarder les appels coûteux à la procédure de recherche. Finalement, nous présentons la procédure UpdateSearchTree qui met à jour l'arbre de recherche entre deux recherches successives.

A* pondérer comme stratégie de recherche

Contrairement à GFRA* qui s'appuie sur une simple recherche de type A*, MGP utilise sa version pondérée. Cette variante de A* surestime volontairement la distance au but en multipliant la valeur retournée par la fonction heuristique d'un coefficient w. La fonction d'évaluation $f(s)$ pour un état s est alors $f(s) = g(s) + w \times h(s)$ où $g(s)$ est le coût pour atteindre s à partir de l'état initial s_0 et $h(s)$ le coût estimé pour atteindre g depuis s. Plus w est élevé, plus la fonction heuristique est prépondérante dans l'exploration de l'espace de recherche. Expérimentalement, il a été démontré que cette stratégie couplée avec une fonction heuristique informative améliore les performances en temps, mais en contrepartie ne garantit plus l'optimaliser de la solution produite (POHL, 1970). Toutefois, cette approche est pertinente pour le problème considéré, car il est plus important de trouver rapidement une solution que de trouver un plan solution optimal. Par conséquent, utiliser une recherche pondérée de type A* peut améliorer de manière significative les performances de MGP (cf. §4.3). En outre, il est important de noter que l'utilisation de cette procédure de recherche n'affecte pas la correction et la complétude de MGP.

La version de A* pondéré utilisée dans notre approche (cf. Algo. 6) est une version légèrement modifiée de l'algorithme classique proposé par (POHL, 1970). Celui-ci prend en entrée un problème de planification (A, s_0, g), un coefficient w et le compteur i qui comptabilise le nombre de recherches effectuées. À chaque état s, l'algorithme associe quatre valeurs : sa valeur $g(s)$ et sa valeur $h(s)$, un pointeur vers son état père, $parent(s)$, dans l'arbre de recherche et l'itération à laquelle il a été créé, $iteration(s)$. $iteration(s)$ permet à l'algorithme de ne mettre à jour que les états nouvellement créés. Au premier appel de la procédure, les listes OPEN et CLOSED contiennent l'état initial s_0 du problème de planification tel que $g(s_0) = 0$ et $h(s_0) = H(s_0, g)$ où H est la fonction heuristique qui estime le coût de s_0 au but g.

La procédure de recherche A* pondéré (cf. Algo. 6) consiste à répéter itérativement les étapes suivantes : (1) sélectionner l'état s de la liste OPEN qui possède la plus petite valeur de $f(s)$ (ligne 2), (2) supprimer l'état s de la liste OPEN (ligne 4), et (3) développer les états fils de s en utilisant la fonction de transition γ. Pour chaque état fils s' obtenu avec la fonction de transition γ, si s' n'est pas dans une des deux listes OPEN et CLOSED, alors A* pondéré calcule $g(s') = g(s) + c(s, s')$ où $c(s, s')$ représente le coût pour atteindre s' depuis s. Si s' n'a pas encore été exploré (lignes 9-13), s' est ajouté à la liste OPEN. Sinon, A* pondéré met à jour les informations relatives à s'. De plus, si A* trouve un plan plus court pour atteindre s' (lignes 15-18), il met alors à jour s' : $g(s') = g(s) + c(s, s')$ et $parent(s') = s$. Finalement, si la valeur heuristique est obsolète (lignes 19-21) parce que le but a changé, A* pondéré met à jour la valeur heuristique de s'.

A* pondéré possède deux conditions d'arrêt : (1) lorsque la liste OPEN est vide (aucun état contenant le but n'a pu être trouvé) et (2) lorsqu'un état contenant le but est trouvé (ligne 3). Dans ce cas, un plan solution peut être extrait de l'arbre de recherche construit par la procédure de recherche (cf. Algorithme. 5, ligne 5).

Stratégies de retardement

Afin de limiter le nombre de recherches effectuées et améliorer ses performances, MGP retarde autant que possible le déclenchement d'une nouvelle recherche lorsque le but évolue. Pour cela, MGP utilise deux stratégies originales mises en œuvre dans la procédure CanDelayNewSearch (cf. Algorithme 5 ligne 4) :

Algorithme 6 : Search(A, s, g, i)

```
1  while open ≠ ∅ do
2      s ← argmin_{s'∈open}(g(s') + w × h(s'))
3      if g ⊆ s then return SUCCESS
4      open ← open \ {s}, closed ← closed ∪ {s}
5      foreach action a ∈ {a' ∈ A | pre(a') ⊆ s} do
6          s' ← γ(s, a), cost ← g(s) + c(s, s')
7          if s' ∉ open ∪ closed then
8              open ← open ∪ {s'}
9              g(s') ← cost
10             h(s') ← H(s', g)
11             parent(s') ← s
12         else
13             if cost < g(s') then
14                 open ← open ∪ {s'}
15                 g(s') ← cost
16                 parent(s') ← s
17             if iteration(s') < i then
18                 h(s') ← H(s', g)
19                 iteration(s') ← i
20 return FAILURE
```

Open Check (OC). MGP vérifie si le nouveau but est encore présent dans l'arbre de recherche, i.e., dans la liste OPEN ou dans la liste CLOSED. Dans ce cas, un nouveau plan peut être extrait directement à partir de l'arbre de recherche du nouvel état but à l'état courant. Contrairement à GFRA* qui ne teste que la liste CLOSED contenant les états explorés, MGP effectue également le test des états feuilles de l'arbre de recherche contenus dans la liste OPEN avant de débuter une nouvelle recherche. Cette vérification limite le nombre de recherches inutiles et les réajustements coûteux de l'arbre de recherche.

Plan Follow (PF). MGP effectue une estimation pour savoir si l'exécution de son plan courant le rapproche du but, même si celui-ci a changé. Chaque fois que le but évolue et avant de lancer une nouvelle recherche, MGP évalue si le nouveau but est proche du but précédent et détermine si le plan courant peut encore être utilisé. Cette évaluation s'appuie sur l'estimation de la fonction heuristique et le calcul d'une comparaison entre l'état courant s, le but précédent p et le nouveau but g :

$$H(s, g) \times c > H(s, p) + H(p, g) \tag{4.1}$$

c est appelé le coefficient de retardement. MGP suit le plan courant tant que l'inégalité 4.1 est vérifiée, i.e., tant qu'il estime préférable d'exécuter le plan courant pour atteindre l'ancien but puis se "diriger" vers le nouveau but plutôt que d'exécuter un plan allant directement de l'état courant s au nouveau but g. Les valeurs de $c > 1$ permettent d'ajuster le délai avant une nouvelle recherche. Étant donné que les recherches sont très coûteuses, les retarder améliore les performances de MGP, mais altère la qualité des plans (cf. §4.3).

Mise à jour incrémentale de l'arbre de recherche

MGP adapte incrémentalement l'arbre de recherche à chaque nouvelle recherche (cf. UpdateTreeSearch Algorithme 5 ligne 13). Contrairement à GFRA* qui met à jour la valeur heuristique de tous les états de l'arbre de recherche en estimant la distance au nouveau but, MGP s'appuie sur une stratégie moins agressive de mise à jour afin de réduire le nombre d'appels à la fonction heuristique et ainsi améliorer ses performances globales.

Dans cet objectif, MGP copie les états de la liste OPEN contenant les états feuilles de l'arbre de recherche dans la liste CLOSED, puis vide la liste OPEN et y ajoute l'état courant en prenant soin de mettre à jour sa valeur heuristique (appel de la fonction heuristique pour estimer la distance de l'état courant au nouveau but). Pour indiquer que la valeur heuristique est à jour, MGP marque l'état avec le compteur incrémenté à chaque nouvelle recherche (cf. Algorithme 6, ligne 20). Chaque fois qu'un état dans la liste CLOSED est rencontré avec une valeur d'itération inférieure à la valeur du compteur d'itération courant, l'état est ajouté dans la liste OPEN et sa valeur heuristique est mise à jour. Cette stratégie possède deux avantages : (1) elle réduit le nombre d'états créés en réutilisant les états précédemment construits et (2) elle réduit de manière significative le temps nécessaire à la mise à jour des valeurs heuristiques des états à partir du nouveau but en se limitant aux états explorés au cours de la nouvelle recherche.

4.3 Expérimentation et évaluation

L'objectif de ces expériences est d'évaluer les performances de MGP en fonction des différentes stratégies de retardement proposées : *Open Check* (OC) et *Plan Follow* (PF) comparativement aux approches existantes. Nous avons choisi de comparer MGP à l'algorithme GFRA* qui est l'algorithme de référence dans le domaine des algorithmes de type MTS et à l'approche naïve SA* (*Successive A**) qui consiste à appeler la procédure de recherche A* à chaque fois que le but évolue. Les benchmarks utilisés pour l'évaluation sont issus des compétitions internationales de planification (IPC). Nous utilisons pour l'ensemble de nos tests la fonction heuristique non admissible proposée par (HOFFMANN et NEBEL, 2001). Dans la suite, nous évaluons six algorithmes : Successive A* (SA*), *Generalized Fringe-Retrieving A** (GFRA*), MGP sans stratégie de retardement (MGP), MGP avec *Open Check* (MGP+OC), MGP avec *Plan Follow* (MGP+PF) et finalement MGP avec les deux stratégies simultanément (MGP+OC+PF).

4.3.1 Simulation et dynamique d'évolution du but

Classiquement, les algorithmes MTS supposent que la cible mouvante – la *proie* – suit toujours le plus court chemin de sa position courante à sa nouvelle position choisie de manière aléatoire parmi les positions libres sur une grille. Chaque fois que la proie a atteint cette position, une nouvelle position est alors sélectionnée de manière aléatoire et le processus se répète. Tous les n déplacements, la proie reste immobile laissant ainsi au *chasseur* une chance de l'attraper. Cette approche n'est malheureusement pas directement transposable à la planification de tâches en boucle fermée. En effet, si l'on applique la même approche dans le cadre plus général de la planification, le nouveau but n'est pas systématiquement atteignable depuis l'ancien.

Afin de s'en approcher au mieux, nous avons décidé de faire évoluer le but à un pas de temps donné en lui appliquant de manière aléatoire une action parmi

les actions définies dans le problème de planification. Nous faisons ici l'hypothèse que le but évolue en accord avec le modèle que l'agent a du monde. Le but que nous faisons évoluer est l'état solution obtenu par le premier appel à la procédure de recherche. En effet, dans le cas général, il peut y avoir plusieurs états but (le but d'un problème de planification de type STRIPS est un état partiel dont l'inclusion dans un état complet est requise). Le processus est répété plusieurs fois pour rendre le but plus difficile à atteindre. Notons que cette façon de faire évoluer le but garantit que le nouveau but est toujours atteignable depuis le but courant. Cependant, il n'est pas garanti que MGP soit capable d'atteindre le but puisque celui-ci peut évoluer bien plus rapidement que MGP ne s'en rapproche.

En outre, il est important de mentionner que notre cadre expérimental est plus difficile que celui utilisé classiquement pour comparer les algorithmes de type MTS. En effet dans notre cas, le but n'évolue pas en fonction des actions exécutées, mais en temps réel en fonction du temps de calcul nécessaire à la génération des plans solution. Autrement dit, plus un algorithme passe de temps à explorer l'espace de recherche du problème qui lui a été confié, plus le but évolue et plus le problème devient difficile à résoudre.

Pour paramétrer l'évolution du but en fonction du temps de recherche, un compteur t est incrémenté chaque fois qu'un état est exploré au cours d'une recherche et chaque fois que la fonction heuristique est appelée pour estimer la distance au but. En effet, ces deux opérations sont de loin les plus coûteuses de MGP. À partir de ce compteur, le nombre n d'actions appliquées au but est calculé comme suit :

$$n = (t - t_p)/g_r \tag{4.2}$$

où t_p représente la précédente valeur du compteur t et g_r le coefficient d'évolution du but. g_r nous permet d'ajuster la vitesse d'évolution du but et la difficulté de l'atteindre indépendamment du temps ou des caractéristiques techniques de la machine physique sur laquelle sont réalisées les expérimentations.

4.3.2 Cadre expérimental

Étant donnée l'évolution aléatoire du but, chaque expérience a été réalisée 100 fois pour un problème et un coefficient d'évolution de but donné pour obtenir des données statistiquement représentatives. Les résultats présentés par la suite sont donc des moyennes. Toutes les expériences ont été menées sur une machine Linux Fedora 16 équipée d'un processeur Intel Xeon 4 Core (2.0Ghz) avec un maximum de 4Go de mémoire vive et 60 secondes de temps CPU allouées pour chaque recherche.

Dans un premier temps, les différents algorithmes sont testés sur le domaine Blockworld extrait de IPC-2 afin de mesurer leurs performances respectives en fonction du coefficient d'évolution du but. Dans un deuxième temps, nous présentons quelques résultats montrant l'impact du coefficient de retardement ainsi que de la pondération de l'heuristique (nous présentons ici seulement les résultats obtenus par les meilleurs algorithmes de la première évaluation). Pour terminer, dans un troisième temps, nous proposons une vue d'ensemble des performances des différents algorithmes sur différents domaines et problèmes.

Les performances des algorithmes sont mesurées par : (1) le pourcentage de succès, i.e., le nombre de fois où un algorithme réussit à atteindre le but, (2) le temps mis pour atteindre le but, (3) la longueur des plans solutions trouvés.

	Temps			Longueurs		
	Moy.	Min.	Max.	Moy.	Min.	Max
SA*	2,49	0,35	5,63	0,20	0,11	0,46
GFRA*	1,64	0,01	5,61	0,13	0,01	0,33
MGP	2,11	0,52	3,37	0,14	0,09	0,25
MGP+OC	0,76	0,30	1,23	0,01	0,00	0,02
MGP+PF	2,38	0,87	5,29	0,20	0,11	0,71
MGP+OC+PF	3,02	1,10	5,95	0,19	0,10	0,59

TABLE 4.1 – Coefficients de variation moyens, minimums et maximums calculés sur le problème Blockworld 20

4.3.3 Comparaison des algorithmes sur Blockworld

Nous présentons une comparaison des six algorithmes précédemment introduits – SA*, GFRA*, MGP, MGP+OC, MGP+PF et MGP+OC+PF – sur le problème 20 du domaine Blockworld de la compétition IPC-2 en fonction du coefficient d'évolution du but ainsi qu'une comparaison des mêmes algorithmes sur tout le domaine Blockworld afin d'en donner une comparaison globale. Le coefficient de retardement pour MGP+PF et MGP+OC+PF est fixé arbitrairement à 1.6 et le coefficient de pondération de l'heuristique est de 1 pour tous les algorithmes. L'étude de l'impact de ces paramètres est disponible section 4.3.4.

Le problème 20 du domaine Blockworld

Les figures 4.1(a), 4.1(b) et 4.1(c) présentent respectivement les résultats obtenus en termes de pourcentages de succès, temps de recherche et longueur des plans trouvés sur le problème 20 extrait du domaine Blockworld. Les coefficients de variation moyens, minimums et maximums [1] des résultats présentés sont donnés à titre indicatif par le tableau 4.1.

En ce qui concerne le critère de succès, le meilleur algorithme est MGP+OC+PF. Même avec un coefficient d'évolution du but extrême $g_r = 1$, MGP+OC+PF parvient à trouver une solution dans 95 % des expériences réalisées. Pour les autres stratégies de retardement, les résultats sont légèrement moins bons, mais restent très satisfaisants puisqu'ils restent au-dessus de 80 % de succès. L'approche naïve SA* nécessite un coefficient d'évolution du but 5 fois plus grand pour obtenir le même pourcentage de succès. GFRA* lui n'atteint pas ce pourcentage de succès même avec un coefficient d'évolution du but 30 fois supérieur. Finalement, GFRA* est largement surclassé par les autres algorithmes.

En termes de temps de recherche, le meilleur algorithme est MGP+OC. Puis viennent ensuite MGP+OC+PF, MGP+PF et MGP. Finalement, nous avons SA* et GFRA* très loin derrière. Il semble que dans le cadre du problème étudié la stratégie OC soit extrêmement efficace, le nouveau but étant souvent présent dans la liste des nœuds pendants de A*. En outre, le couplage des deux stratégies de retardement

1. En théorie des probabilités et statistiques, le coefficient de variation, également appelé écart relatif, est une mesure de dispersion relative : il se calcule comme le rapport entre l'écart type et la moyenne. Ce nombre est sans unité ; c'est une des raisons pour lesquelles il est préféré à la variance pour traiter des grandeurs physiques. Il nous permet ici de caractériser la qualité des résultats obtenus étant donné le caractère aléatoire de l'évolution du but.

	Temps			Longueurs		
	Moy.	Min.	Max.	Moy.	Min.	Max
SA*	2,52	1,00	5,44	0,30	0,00	0,18
GFRA*	2,07	0,00	5,67	0,33	0,00	0,69
MGP	1,56	0,88	3,10	0,21	0,08	0,56
MGP+OC	1,23	0,23	3,08	0,02	0,00	0,09
MGP+PF	1,86	0,00	3,45	0,22	0,00	0,50
MGP+OC+PF	1,92	0,00	4,10	0,24	0,00	0,61

TABLE 4.2 – Coefficients de variation moyens, minimums et maximums calculés sur le domaine Blockworld

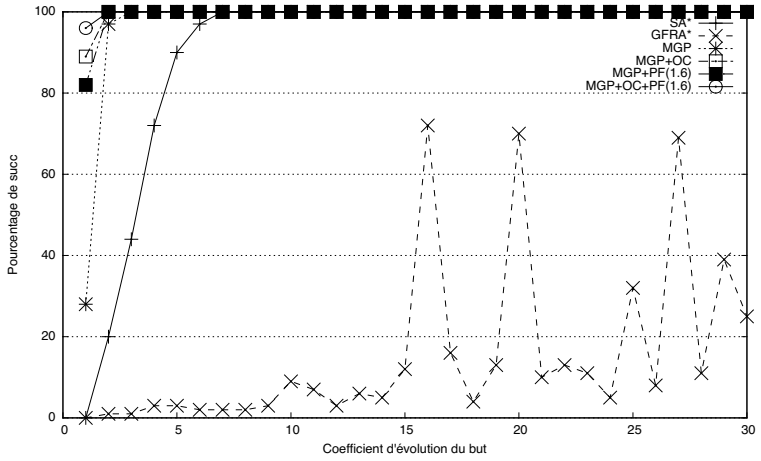
améliore de manière significative la version naïve de MGP. On constate également que les variantes de MGP avec l'option *Plan Follow* ont tendance à avoir un coefficient de variation plus grand. Ceci s'explique en partie par le comportement optimiste de ces variantes et le coût plus important à payer lorsque celles-ci effectuent un mauvais choix.

S'agissant de la longueur des plans trouvés, MGP et ses différentes variantes produisent des plans plus longs que SA* et GFRA* (presque une fois et demie plus longs dans le cas extrême où $g_r = 1$ dans le cas de MGP+OC). Deux raisons expliquent cette différence. Tout d'abord, rappelons que la stratégie de retardement OC vérifie si le nouveau but est déjà présent dans l'arbre de recherche. Si c'est le cas, MGP extrait alors directement un nouveau plan à partir de l'arbre de recherche. Toutefois, si ce mécanisme limite le nombre d'appels à la procédure de recherche, il ne garantit pas que le plan extrait soit optimal. En effet, même si l'heuristique est admissible, l'arbre de recherche n'a pas été construit pour le nouveau but. Deuxièmement, la stratégie de retardement PF (cf. figure 4.1(c)) tend à augmenter la longueur des plans trouvés. La différence de qualité des plans trouvés compense en partie des temps de recherche et des pourcentages de succès bien meilleurs. Cela dépend bien évidemment du critère à optimiser. En outre, les coefficients de variation observés sur la longueur des plans restent faibles, quels que soient les algorithmes.

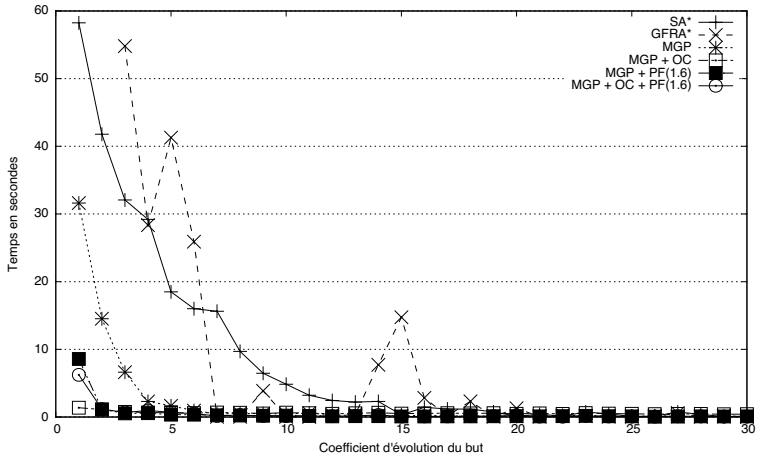
Le domaine Blockworld

Les figures 4.2(a), 4.2(b) et 4.2(c) présentent respectivement les résultats obtenus en termes de pourcentages de succès, temps de recherche et longueurs des plans trouvés sur tous les problèmes du domaine Blockworld. Les problèmes sont classés par ordre de difficulté en fonction du nombre d'états qu'ils peuvent générer. Pour cette série d'expériences, le coefficient d'évolution du but a été fixé à 5. Les autres paramètres restent identiques : chaque expérience est répétée 100 fois et 60 secondes sont allouées pour chaque expérience. Les coefficients de variation moyens, minimums et maximums des résultats présentés sont donnés à titre indicatif par le tableau 4.2.

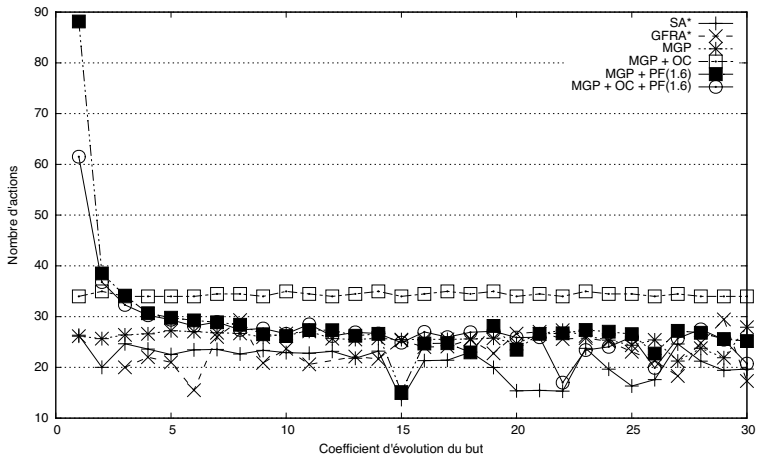
En ce qui concerne le pourcentage de succès, nous retrouvons les résultats obtenus sur le problème 20. GFRA* est largement distancé. Son pourcentage de succès chute de manière drastique dès le problème 16 à 20 %. Puis vient ensuite SA* qui atteint un peu plus des 40 % au problème 21. Pour finir viennent les différentes versions de MGP dont les pourcentages de succès commencent à chuter à partir



(a) Pourcentage de succès en fonction du coefficient d'évolution du but



(b) Temps de recherche en fonction du coefficient d'évolution du but



(c) Longueurs des plans en fonction du coefficient d'évolution du but

FIGURE 4.1 – Analyse comparée des différentes stratégies de MGP sur le problème 20 du domaine Blockworld

des problèmes 22 et 23. On peut toutefois noter un léger avantage à MGP+OC+PF qui obtient près de 90 % de succès sur le problème 24 contre un peu moins de 80 % pour MGP+PF, un peu moins de 70 % pour MGP+OC et 50 % pour MGP.

Étant donné que le temps de recherche est fixe, il arrive un moment où il n'est plus possible de résoudre les problèmes dans le temps imparti. Dans le cadre de notre expérimentation, cette limite se situe au niveau de la plage de problèmes de 22 à 24.

Ces résultats se retrouvent également sur les temps de recherche. Les différentes variantes de MGP surclassent SA* et GFRA* avec des temps de recherche inférieurs à 5 secondes. On retrouve ici, comme pour le problème 20, un léger avantage pour MGP+OC.

Pour terminer, en termes de taille de plans, les longueurs des plans trouvés sont assez semblables. Toutefois, les résultats obtenus sur l'ensemble du domaine confirment une nouvelle fois les résultats obtenus sur le problème 20. SA* et GFRA* trouvent des plans plus courts. Viennent ensuite les variantes de MGP avec un avantage pour MGP, et MGP+OC+PF.

4.3.4 Impact du coefficient de retardement et de l'heuristique pondérée

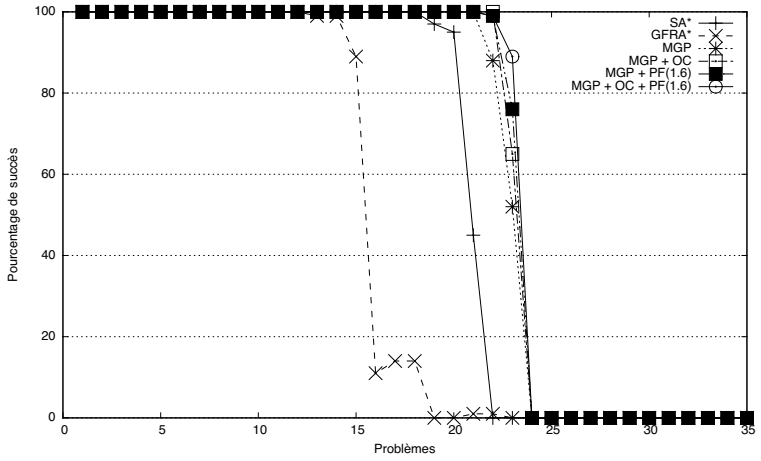
Nous évaluons l'impact du coefficient de retardement ainsi que de la pondération de l'heuristique sur le meilleur algorithme obtenu précédemment, i.e., MGP+OC+PF. Les expériences suivantes ont été réalisées sur le même problème (Blockworld problème 20) pour permettre une comparaison des résultats. Étant donné que MGP+OC+PF a un pourcentage de succès proche de 100 % nous ne présentons dans ce qui suit que les résultats en termes de temps de recherche et de longueur de plans.

Impact du coefficient de retardement

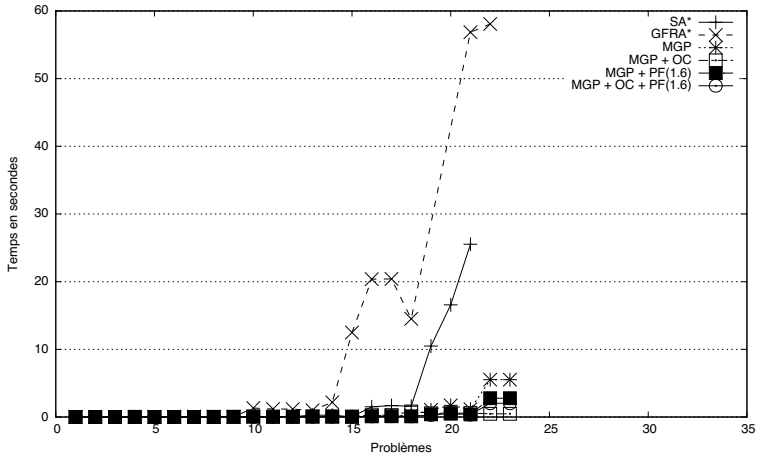
Les figures 4.3(a), 4.3(b), 4.4(a) et 4.4(b) présentent respectivement le temps de recherche et la longueur des plans obtenus en fonction du coefficient d'évolution du but pour les variantes de MGP, MGP+PF et MGP+OC+PF. Nous pouvons faire quatre observations. Premièrement, nous pouvons constater que le coefficient de retardement améliore de manière significative les performances de MGP. Par exemple, MGP+OC+PF (cf. figure 4.4(a)) avec un coefficient de retardement de 2 est 6 fois plus rapide que MGP avec un coefficient de retardement de 0 lorsque l'évolution du but est très rapide ($g_r = 1$). Deuxièmement, on peut constater que le temps de recherche ainsi que la longueur des plans convergent rapidement vers une même valeur, et ce quelle que soit par la suite la valeur de g_r. Par exemple, avec un coefficient $g_r \geq 4$, le coefficient de retardement n'a quasiment plus d'impact sur le temps de recherche et sur la longueur des plans. Troisièmement, on constate qu'une augmentation du coefficient de retardement implique également une augmentation de la longueur des plans trouvés, mais réduit le temps de recherche. Par conséquent, le coefficient de retardement doit être choisi de manière à être un compromis entre la longueur des plans trouvés et temps de recherche. Quatrièmement, on retrouve dans ces courbes les meilleures performances de MGP+OC+PF relativement à MGP+PF en termes de temps de recherche. Toutefois, cet avantage a tendance à s'atténuer avec l'augmentation du coefficient de retardement.

Impact de l'heuristique pondérée

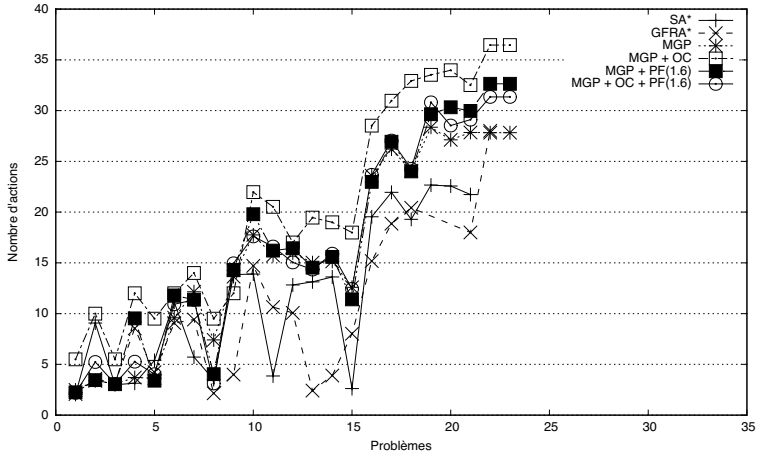
La figure 4.5 présente respectivement le temps de recherche et la longueur des plans obtenus en fonction de la pondération de l'heuristique des différentes variantes de MGP. On constate que la pondération de l'heuristique w améliore, comme le coefficient de retardement, de manière significative les performances de



(a) Pourcentage de succès en fonction de la difficulté des problèmes



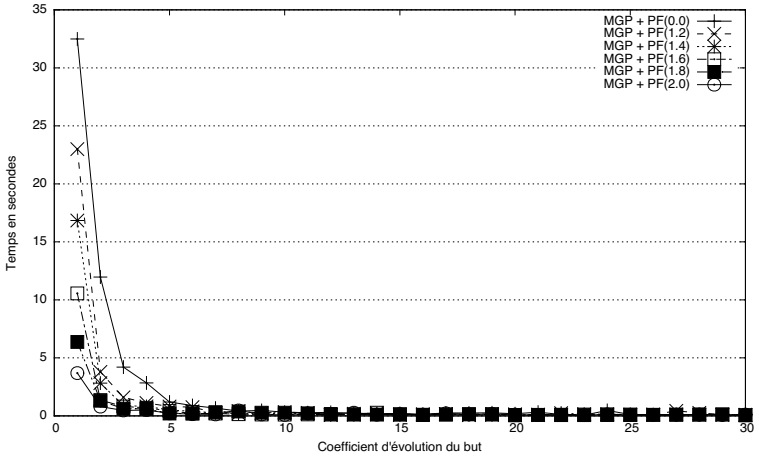
(b) Temps de recherche en fonction de la difficulté des problèmes



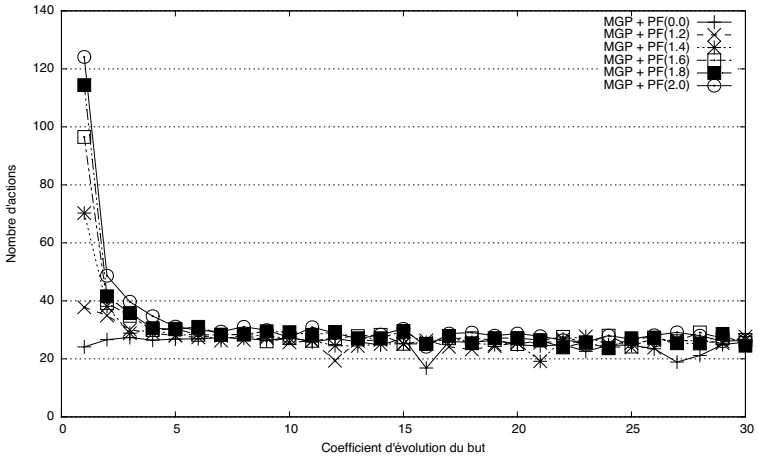
(c) Longueurs des plans en fonction de la difficulté des problèmes

FIGURE 4.2 – Analyse comparée des différentes stratégies de MGP sur le domaine Blockworld

MGP (MGP est 4 fois plus rapide avec $w = 2.0$ qu'avec $w = 1.0$ pour un coefficient d'évolution du but $g_r = 1$). L'augmentation est aussi sensible pour des plus grandes valeurs de g_r même si nous ne le voyons pas au travers des expériences



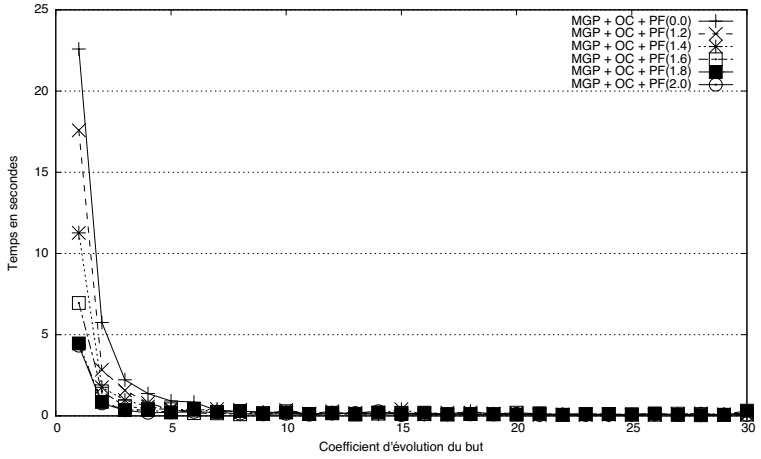
(a) Temps de recherche en fonction du coefficient de retardement et du coefficient d'évolution du but



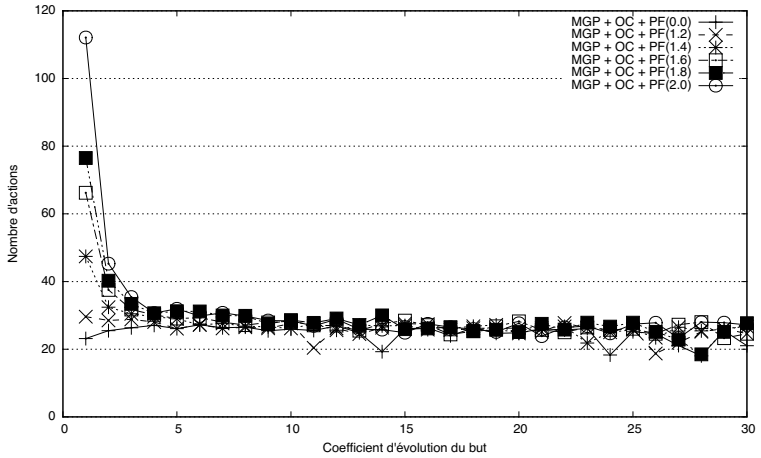
(b) Longueur des plans en fonction du coefficient de retardement et du coefficient d'évolution du but

FIGURE 4.3 – Impact du coefficient de retardement sur les performances de MGP sur le problème 20 du domaine Blockworld sans la stratégie *Open Check*

présentées ici. En outre, l'impact de w sur la longueur des plans obtenus n'est pas significatif : quelle que soit la pondération de l'heuristique, la longueur des plans est comparable pour une valeur de coefficient d'évolution de but donné. En d'autres termes, la pondération de l'heuristique augmente les performances de MGP comme on pouvait s'y attendre, mais ne détériore pas de manière importante la qualité des plans trouvés. En effet, habituellement, l'utilisation d'une heuristique pondérée détériore bien plus la qualité des plans obtenus. Ceci peut s'expliquer par le fait que dans le cadre de notre approche, il est plus important de trouver un plan solution pour donner la direction à suivre pour atteindre le but qui évolue au cours du temps, que de trouver systématiquement un plan solution optimal.



(a) Temps de recherche en fonction du coefficient de retardement et du coefficient d'évolution du but



(b) Longueur des plans en fonction du coefficient de retardement et du coefficient d'évolution du but

FIGURE 4.4 – Impact du coefficient de retardement sur les performances de MGP sur le problème 20 du domaine Blockworld avec la stratégie *Open Check*

4.3.5 Vue d'ensemble des performances sur différents problèmes

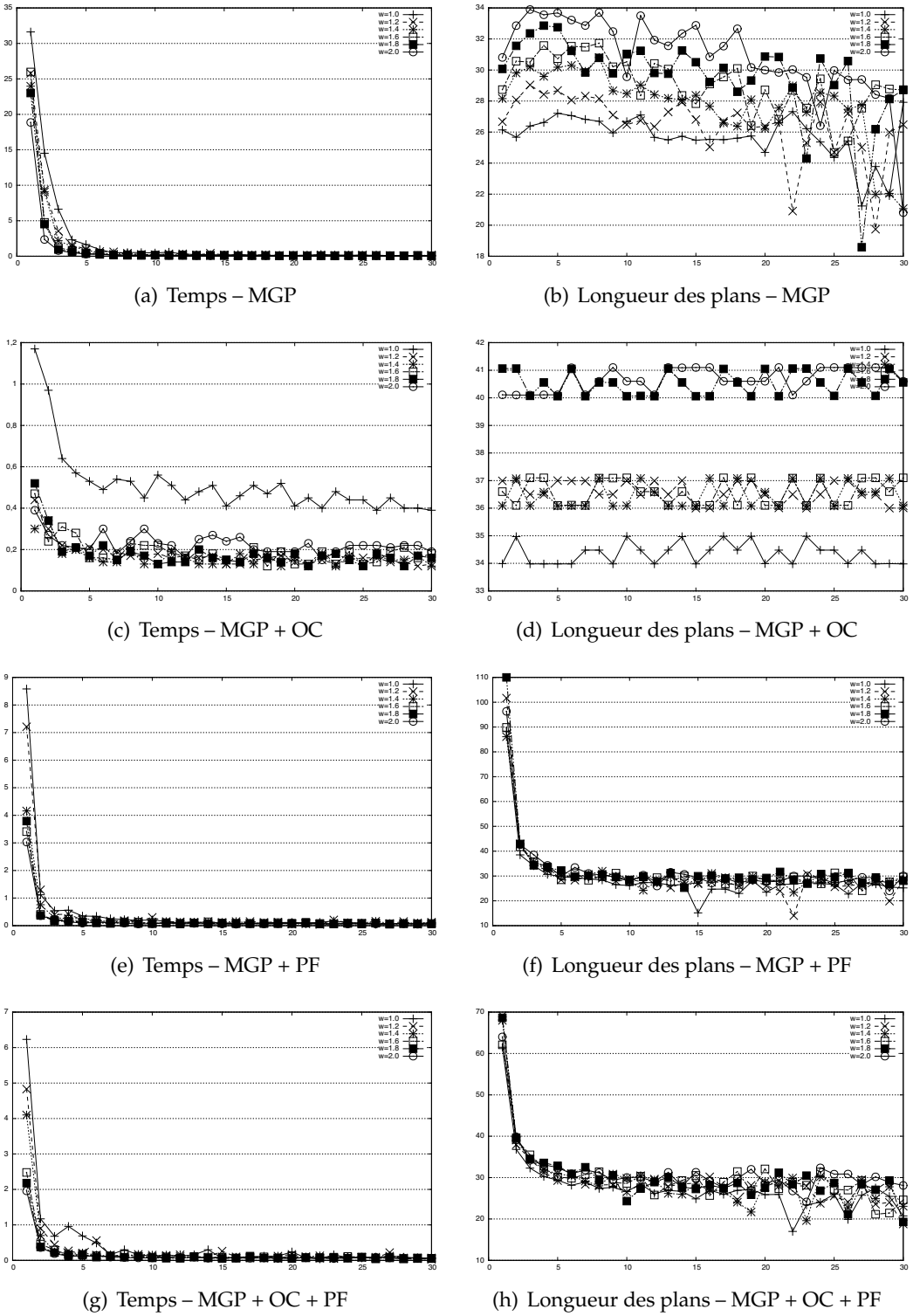
Dans cette partie, nous présentons une vue d'ensemble des performances des différents algorithmes sur plusieurs problèmes issus des compétitions IPC : le tableau 4.3 donne les temps de recherche obtenus, le tableau 4.5 les pourcentages de succès et le tableau 4.4 les longueurs des plans trouvés. Comme précédemment, chaque expérience a été répétée 100 fois pour obtenir des résultats statistiquement représentatifs. À chaque expérience, un temps CPU de 60 secondes et un maximum de 4Go de mémoire vive ont été alloués. Le coefficient de retardement a été fixé à 1.6. Pour tous les tests, nous avons également fixé le coefficient de changement de but à 100. Contrairement au domaine simple Blockworld, nous avons choisi ici un coefficient de changement de but plus grand pour aborder des problèmes plus complexes. Nous avons lancé l'expérimentation sur tous les problèmes des domaines décrits dans les tableaux en suivant la démarche expérimentale présentée en détail sur le domaine Blockworld (cf. §4.3.3). Toutefois, nous ne montrons ici que les résultats obtenus sur les problèmes pour lesquels on observe une inflexion des performances.

Problème	SA*	GFRA*	MGP	OC	PF	OC+PF
airport p16	3,38	0,91	0,20	0,18	0,17	0,12
airport p19	-	13,48	0,40	0,36	0,37	0,21
depot p03	8,12	-	2,40	1,28	1,67	0,78
depot p07	-	-	7,41	6,42	1,73	1,13
driverlog p03	0,03	0,02	0,33	0,22	0,18	0,17
driverlog p06	13,57	-	4,32	5,50	5,53	4,35
elevator p30	24,85	23,69	29,32	3,23	2,22	1,46
elevator p35	23,04	-	-	44,60	35,31	20,80
freecell p20	10,94	-	-	4,72	6,20	3,65
freecell p26	48,76	-	56,61	26,70	32,12	24,20
openstack p06	55,58	55,80	55,20	54,03	48,69	36,30
openstack p07	54,55	57,51	57,35	50,33	46,13	43,32
pipeworld p04	0,50	17,73	1,68	0,49	0,55	0,48
pipeworld p08	-	-	18,02	13,89	13,70	12,51
pathway p02	15,60	9,52	6,92	3,53	2,83	1,74
pathway p04	-	-	-	-	8,05	4,39
rover p03	23,35	14,17	3,72	2,85	2,15	1,87
rover p07	-	-	-	23,51	-	22,54
satellite p03	-	17,26	9,36	3,67	4,56	3,18
satellite p06	-	-	-	-	-	8,97

TABLE 4.3 – Comparaison du temps de recherche des différentes stratégies de MGP sur plusieurs domaines

En termes de temps de recherche, on retrouve la plupart des résultats obtenus sur le problème Blockworld 20 et le domaine Blockworld : MGP+OC+PF est largement plus rapide que les autres algorithmes. Toutefois, on constate que les bonnes performances observées par MGP+OC dans le domaine blockworld ne se vérifient pas sur les autres domaines testés. De plus, MGP+OC+PF surclasse les variantes de MGP avec une seule stratégie de retardement. De même, MGP+OC+PF surclasse les autres algorithmes en termes de pourcentage de succès. Cependant, on peut constater que MGP sans stratégie de retardement échoue sur certains problèmes (Elevator P35 and Freecell P26) que SA* résout, même si d'une manière générale, la version naïve de MGP reste meilleure que SA* et GFRA*. Finalement, en termes de longueur de plans, toutes les approches trouvent des plans de longueurs comparables ce qui ne constitue par conséquent pas un critère discriminant entre les différentes approches.

Pour synthétiser ces résultats, OC et PF améliorent individuellement les performances de MGP qui surclasse SA* et GFRA*. Les deux stratégies de retardement OC+PF couplées donnent de meilleurs résultats que lorsqu'elles sont utilisées séparément. Les résultats obtenus sur différents domaines et problèmes confirment que MGP+OC+PF est le meilleur algorithme et que GFRA* est largement dépassé (GFRA* met à jour la fonction heuristique de tous les états de l'arbre de recherche avant chaque nouvelle recherche ce qui le pénalise comparativement aux autres algorithmes).



(a) Temps – MGP

(b) Longueur des plans – MGP

(c) Temps – MGP + OC

(d) Longueur des plans – MGP + OC

(e) Temps – MGP + PF

(f) Longueur des plans – MGP + PF

(g) Temps – MGP + OC + PF

(h) Longueur des plans – MGP + OC + PF

FIGURE 4.5 – Impact de la pondération de l'heuristique sur les performances de MGP sur le problème 20 du domaine Blockworld

4.4 Conclusion

Nous avons proposé une approche pour la planification de tâches en boucle fermée, appelée MGP, qui adapte continuellement son plan aux évolutions successives du but. MGP s'appuie sur une recherche heuristique pondérée incrémentale

Problème	SA*	GFRA*	MGP	OC	PF	OC+PF
airport p16	80,98	80,98	79,15	81,25	80,81	81,12
airport p19	-	91,98	90,12	91,52	90,62	91,14
depot p03	20,80	-	21,73	21,67	25,18	22,83
depot p07	-	-	25,24	23,08	26,00	24,74
driverlog p03	7,30	4,14	8,42	11,57	12,57	9,57
driverlog p06	12,00	-	14,00	15,00	11,23	16,91
elevator p30	29,20	27,88	27,33	28,42	27,74	28,80
elevator p35	34,00	-	-	33,05	32,20	32,18
freecell p20	29,97	-	-	29,99	29,99	29,99
freecell p26	37,02	-	38,00	37,01	37,01	37,01
openstack p06	50,41	51,13	51,28	50,68	50,06	50,54
openstack p07	51,12	52,14	50,76	51,25	50,72	51,02
pipeworld p04	3,21	11,86	7,61	9,19	10,20	8,12
pipeworld p08	-	-	18,21	21,09	19,76	21,02
pathway p02	27,18	26,39	26,27	26,93	37,02	40,76
pathway p04	-	-	-	-	34,92	35,12
rover p03	44,53	44,73	44,40	44,54	44,66	44,24
rover p07	-	-	-	43,20	-	43,50
satellite p03	-	42,00	26,00	24,69	32,12	25,80
satellite p06	-	-	-	-	-	26,00

TABLE 4.4 – Comparaison de la longueur des plans obtenus des différentes stratégies de MGP sur plusieurs domaines

de type A* et entrelace planification et exécution. Afin de limiter le nombre de recherches effectuées pour prendre en compte les changements dynamiques du but au cours du temps, MGP retarde autant que possible le déclenchement de nouvelles recherches. Pour cela, MGP utilise deux stratégies de retardement : *Open Check* (OC) qui vérifie si le but est encore présent dans l'arbre de recherche et *Plan Follow* (PF) qui estime s'il est préférable d'exécuter les actions du plan courant pour se rapprocher du nouveau but plutôt que de relancer une nouvelle recherche. De plus, MGP utilise une stratégie conservatrice et efficace de mise à jour incrémentale de l'arbre de recherche lui permettant de réduire le nombre d'appels à la fonction heuristique et d'accélérer la recherche d'un plan solution.

Nous avons montré expérimentalement que MGP surclassait l'approche naïve SA* et l'algorithme de référence GFRA*. Nous avons également montré que : (1) la combinaison des deux stratégies de retardement OC+PF donnait de meilleurs résultats que chacune considérée individuellement ; (2) le coefficient de retardement qui permet de paramétrer la stratégie PF doit être un compromis entre la longueur des plans trouvés et le temps de recherche et (3) la pondération de l'heuristique améliore de manière significative les performances de MGP sans trop détériorer la qualité des plans trouvés.

Ce travail ouvre la voie à plusieurs pistes de recherche. Dans un premier temps, il serait intéressant de complexifier notre cadre expérimental. Nous supposons en effet que le but évolue de manière aléatoire en fonction du temps qui s'écoule. Même si cette hypothèse de travail est plus réaliste que celle utilisée classiquement pour comparer des algorithmes de type MTS, elle reste restrictive puisque le but évolue de proche en proche sans stratégie particulière. Pour relaxer cette hypothèse, nous envisageons de guider l'évolution du but en utilisant une fonction heuristique. Cette

Problème	SA*	GFRA*	MGP	OC	PF	OC+PF
airport p16	97	100	100	100	100	100
airport p19	-	72	81	100	100	100
depot p03	10	-	99	30	60	100
depot p07	-	-	33	12	14	88
driverlog p03	100	100	100	100	100	100
driverlog p06	1	-	1	56	88	98
elevator p30	69	99	91	100	100	100
elevator p35	1	-	-	19	5	55
freecell p20	39	-	-	99	100	100
freecell p26	56	-	1	100	100	100
openstack p06	77	48	62	70	96	99
openstack p07	44	15	21	100	99	99
pipeworld p04	99	7	99	99	98	100
pipeworld p08	-	-	48	70	60	76
pathway p02	100	100	100	100	100	100
pathway p04	-	-	-	-	22	48
rover p03	48	8	99	99	94	99
rover p07	-	-	-	32	-	52
satellite p03	-	1	4	16	15	52
satellite p06	-	-	-	-	-	28

TABLE 4.5 – Comparaison du pourcentage de succès des différentes stratégies de MGP sur plusieurs domaines

approche nous permettrait alors de caractériser plus précisément la difficulté d'atteindre un but et donc les problèmes à résoudre. De plus, un prolongement naturel de ce travail consiste à l'étendre dans le cadre d'une planification en temps réel telle présentée dans les deux premiers chapitres de ce manuscrit.

Deuxième partie

Planifier à plusieurs

NOUS abordons dans cette partie la problématique de la distribution du processus de planification. Autrement dit, comment plusieurs systèmes autonomes peuvent-ils réaliser de manière distribuée la synthèse d'un plan d'actions pour atteindre un objectif commun ?

Il est important de noter que nous ne nous intéressons pas ici aux approches qui consistent à déléguer à un seul agent la responsabilité de planifier pour l'ensemble des autres agents. En effet, dans ce cas, seule l'exécution des plans est distribuée. Il est alors possible d'utiliser des techniques classiques de planification, notamment celles présentées dans les chapitres précédemment. Bien que ces approches soient tout à fait réalistes dans de nombreux contextes applicatifs, d'un point de vue pratique, la distribution croissante des applications pour lesquelles il n'est pas possible d'envisager un contrôle centralisé, pour des raisons organisationnelles ou techniques, motive la conception de mécanismes de planification où non seulement l'exécution, mais aussi la génération du plan lui-même est distribuée. Ceci implique que les agents soient capables de se concerter pour décomposer la tâche collective en un ensemble de sous-tâches, de raisonner sur les capacités des autres agents ou encore d'élaborer une vue commune de l'état de l'environnement. L'enjeu ici n'est pas une *simple* coordination des activités des agents partageant un même environnement, mais la co-construction d'un plan pour atteindre un objectif commun.

Il est facile de définir la planification distribuée par extension de son modèle centralisé. Elle peut se définir alors comme un processus distribué qui prend en entrée : une modélisation des actions réalisables par les agents impliqués dans le processus de planification ; une description de l'état du monde connu de ceux-ci et un objectif commun. Le processus doit alors retourner un ensemble organisé d'actions, i.e., un plan, la plupart du temps exprimant une certaine forme de concurrence entres les actions, qui, s'il est exécuté, permet d'atteindre l'objectif. Une telle définition permet facilement d'appréhender ce qu'est la planification distribuée, mais cache la complexité et le découpage des différentes phases du processus. La littérature distingue généralement cinq phases (cf. figure 4.6) :

1. *Une phase de décomposition.* Les agents raffinent l'objectif initial en sous-objectifs atteignables individuellement par les agents. Les approches proposées reposent le plus souvent sur des techniques centralisées de planification, e.g., HTN (YOUNES et SIMMONS, 2003) ou sur les graphes de planification (IWEN et MALI, 2002).
2. *Une phase de délégation.* Les agents s'assignent mutuellement les objectifs obtenus lors de la première phase. Cette étape s'appuie sur des techniques d'allocation de ressources dans un contexte centralisé (BAR-NOY et al., 2001) ou dans un contexte distribué basées par exemple sur la négociation (ZLOTKIN et ROSENSCHEIN, 1990), l'argumentation (TAMBE et JUNG, 1999), ou encore la synchronisation (CLEMENT et BARRETT, 2003).

3. *Une phase de planification locale.* Au cours de cette phase, chaque agent essaie de trouver un plan individuel pour atteindre l'objectif qui lui est assigné à l'étape 2. Les techniques utilisées reposent sur les algorithmes classiques de planification mono agent, e.g., (BLUM et FURST, 1997 ; NAU et al., 2003 ; PENBERTHY et WELD, 1992).
4. *Une phase de coordination.* Lors de cette phase, chaque agent vérifie que le plan local élaboré à l'étape précédente n'entre pas en conflit avec les plans locaux des autres agents. Contrairement à la planification mono agent, cette étape est nécessaire pour garantir l'intégrité fonctionnelle du système, i.e., assurer que le but de chaque agent reste atteignable dans le contexte global. La problématique de la coordination a été tout particulièrement étudiée dans le domaine des systèmes multi-agents notamment en la considérant comme un processus de fusion de plans (TONINO et al., 2002) ou de résolution de conflits (COX et DURFEE, 2005).
5. *Une phase d'exécution.* Finalement, le plan exempt de conflit est exécuté par les agents.

Bien que cette décomposition soit conceptuellement correcte, l'expérience montre que les cinq phases sont souvent entrelacées : (*i*) les phases 1 et 2 sont souvent fusionnées à cause du lien fort qu'il existe entre les capacités des agents et la décomposition de l'objectif ; (*ii*) la dernière phase d'exécution conduit presque toujours à une replanification (FOX et al., 2006) due à un événement non prévu lors de l'élaboration du plan et nécessite un retour aux phases 1, 2 et 3 ; (*iii*) la phase de coordination peut être réalisée avant (SHOHAM et TENNENHOLTZ, 1995), après (TONINO et al., 2002) ou pendant (LESSER et al., 2004) l'étape de planification locale. Ces remarques soulignent le manque de modèles capables d'embrasser les différentes phases du processus de planification distribuée.

Dans la suite de cette partie, nous allons présenter trois de nos contributions qui adressent cette problématique. Pour chacune de ces contributions, nous ferons l'hypothèse qu'aucun des agents ne peut résoudre le but commun qui leur est assigné seul. Les agents sont donc dans l'obligation de coopérer en partageant leurs compétences et leurs connaissances.

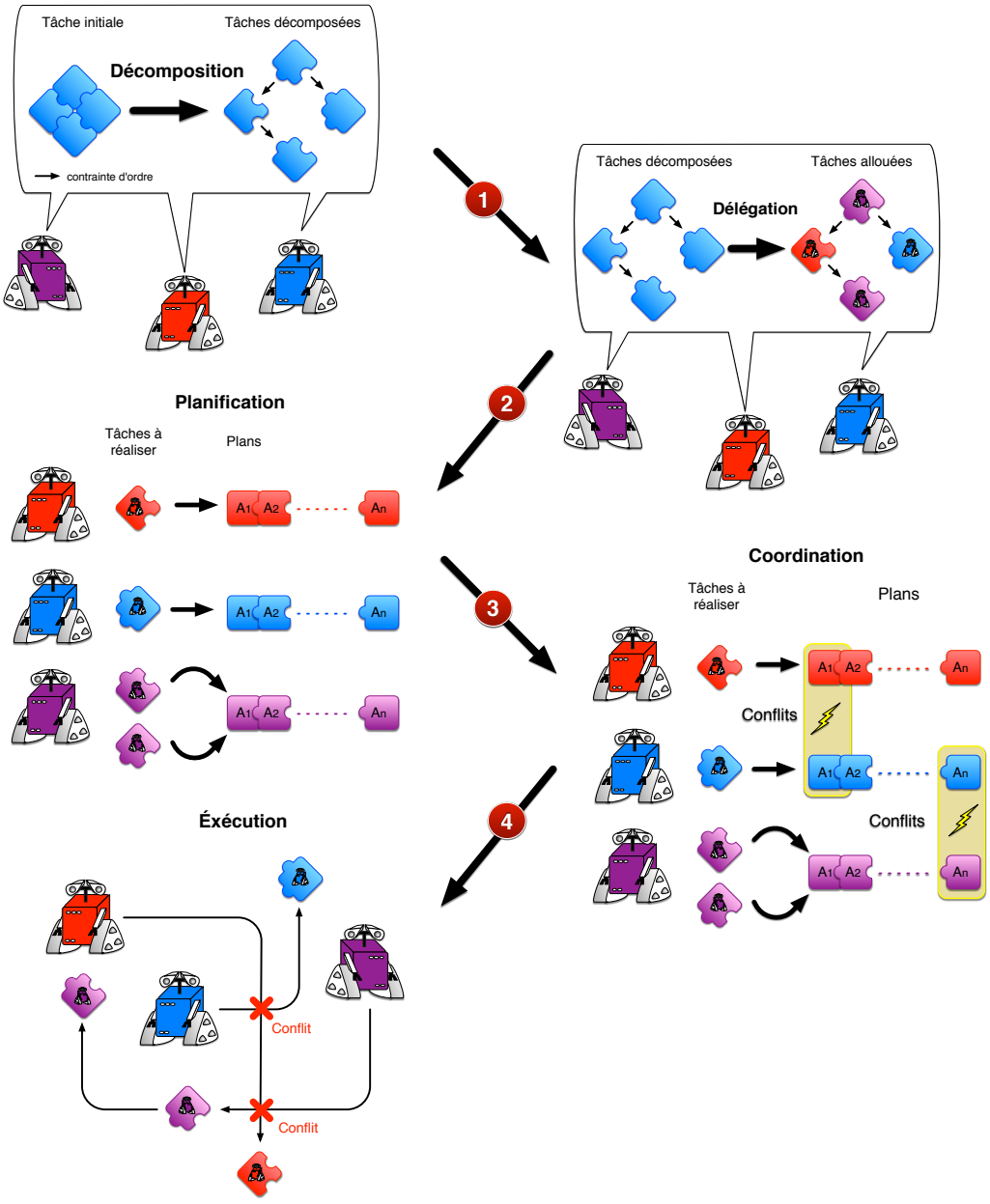


FIGURE 4.6 – Planification distribuée

Chapitre 5

Planifier à plusieurs par fusion incrémentale de graphes de planification

POUR tenter de répondre en partie à la critique formulée dans le préambule de cette partie, nous présentons ici une première de nos contributions (PELLIER, 2010) dans le cadre la planification distribuée dans laquelle l'élaboration d'un plan partagé est vue comme un processus collaboratif et itératif, intégrant les quatre premières étapes décrites précédemment. L'idée forte de notre contribution consiste à amalgamer au plus tôt, i.e., au sein du processus local de planification, la phase de coordination en s'appuyant sur une structure classique dans le domaine de planification, les graphes de planification (BLUM et FURST, 1997). Cette structure a été utilisée pour la première fois par le planificateur Graphplan. Elle possède trois principaux avantages : elle est calculable en temps polynomial ; elle permet d'utiliser les nombreux travaux portant sur la recherche d'un plan solution dans les graphes de planification ; finalement, elle est assez générique pour représenter une certaine forme de concurrence entre les actions. Ce dernier point est un atout majeur dans le cadre de la planification multi-agent.

L'intégration des phases de coordination et de planification locale, au sein d'un même processus que nous appelons *fusions incrémentales de graphes*, présente l'avantage de limiter les interactions négatives entre les plans individuels des agents, mais aussi, et surtout, de prendre en compte les interactions positives, i.e., d'aide ou d'assistance entre agents. Prenons le temps d'un court exemple pour illustrer l'importance de considérer les interactions positives dans un contexte distribué. Supposons un instant que le but d'un agent soit de prendre un objet dans une pièce dont la porte est fermée à clé. Même si l'agent possède un plan pour atteindre son but une fois la porte ouverte, mais qu'il n'est pas capable d'ouvrir la porte, le processus de planification échoue. Supposons maintenant qu'un second agent soit capable d'ouvrir la porte, il existe alors un plan solution. Faut-il encore que le premier agent soit en mesure de considérer, au moment de la synthèse de son plan local, l'aide que peut lui apporter le second agent, et que ce dernier soit en mesure de proposer l'action requise ? Si l'on comprend facilement que l'identification des interactions négatives est nécessaire pour éviter les conflits entre les agents, on voit bien également au travers de cet exemple la nécessité d'identifier également les interactions positives puisqu'elles constituent la base sur laquelle les agents peuvent coopérer.

Dans la suite, nous présentons tout d'abord les définitions préliminaires nécessaires à la formalisation de notre contribution, l'algorithme DPGM (*Distributed Planning through Graph Merging*), puis nous en détaillons les cinq phases : (1) la décomposition du but global en buts individuels ; (2) l'expansion et la fusion des

graphes de planification ; (3) la fusion des graphes de planification ; (4) l'extraction d'un plan individuel et finalement (5) la coordination des plans individuels solution. Finalement, nous terminons en proposant une évaluation de l'approche proposée.

5.1 Compléments de notation et définitions préliminaires

Les opérateurs de planification qui traduisent les compétences des agents dans notre approche sont définis comme des fonctions de transition au sens STRIPS comme défini §1.4.

5.1.1 Indépendance entre actions et graphe de planification

Commençons par définir le concept d'indépendance entre deux actions. L'indépendance entre actions n'est pas une propriété spécifique à un problème de planification donné. Elle découle uniquement de la description des opérateurs des agents et traduit le fait que deux actions peuvent s'exécuter de manière concurrente indépendamment du contexte au moment de leur exécution.

Définition 5.1 (Indépendance)**.** *Deux actions a et b sont* indépendantes *ssi $\mathit{effects}^-(a) \cap (\mathit{precond}(b) \cup \mathit{effects}^+(b)) = \emptyset$ et $\mathit{effects}^-(b) \cap (\mathit{precond}(a) \cup \mathit{effects}^+(a)) = \emptyset$. Par extension un ensemble A d'actions est indépendant si toutes les paires d'actions de A sont indépendantes deux à deux.*

Poursuivons en définissant formellement ce qu'est un graphe de planification (BLUM et FURST, 1997). Des exemples sont visibles figure 5.3.

Définition 5.2 (Graphe de planification)**.** *Un* graphe de planification *G est un graphe orienté par niveaux*[1] *organisés en une séquence alternée de propositions et d'actions de la forme $\langle P_0, A_0, P_1, \ldots, A_{i-1}, P_i \rangle$. P_0 contient les propositions de l'état initial. Les niveaux A_i avec $i > 0$ sont composés de l'ensemble des actions applicables à partir des niveaux propositionnels P_i et les niveaux P_{i+1} sont constitués des propositions produites par les actions de A_i. Les arcs sont définis de la manière suivante : pour toutes les actions $a \in A_i$, un arc relie a avec toutes les propositions $p \in P_i$ qui représentent les préconditions de a ; pour toutes les propositions $p \in P_{i+1}$ un arc relie p aux actions $a \in A_i$ qui produisent ou suppriment p. À chaque niveau, le maintien des propositions du niveau i au niveau $i+1$ est assuré par des actions appelées* no-op *qui ont une unique proposition comme précondition et effet positif.*

Définissons maintenant le concept d'exclusion mutuelle entre deux actions en prenant en compte leur contexte d'exécution.

Définition 5.3 (Exclusion mutuelle)**.** *Deux actions a et b sont* mutuellement exclusives *ou mutex ssi a et b sont dépendantes ou si une précondition de a est mutuellement exclusive avec une précondition de b. Deux propositions p et q d'un niveau propositionnel P_i d'un graphe de planification sont mutuellement exclusives si toutes les actions de A_i avec p comme effet positif sont mutuellement exclusives avec toutes les actions qui produisent q au même niveau, et s'il n'existe aucune action de A_i qui ne produit à la fois p et q. Par la suite, nous noterons l'ensemble des exclusions mutuelles d'un niveau A_i et P_i respectivement μA_i et μP_i.*

1. Un graphe dont les sommets sont partitionnés en ensembles disjoints tels que les arcs connectent seulement les sommets des niveaux adjacents.

Nous dirons qu'un graphe de planification G de niveau i a atteint son *point fixe* si $\forall i, i > j$, le niveau i de G est identique au niveau j, i.e., $P_i = P_j$, $\mu P_i = \mu P_j$, $A_i = A_j$ et $\mu A_i = \mu A_i$. Il est important de noter que tout graphe de planification a un point fixe.

5.1.2 Agent, problème et plan solution

Commençons par définir ce qu'est un agent dans notre modèle.

Définition 5.4 (Agent)**.** *Un* agent α *est un couple* (O, s_0) *où* O *définit l'ensemble des opérateurs que peut réaliser* α *et* s_0 *ses croyances initiales sur le monde.*

Nous introduisons par extension $\mathit{precond}(\alpha)$ qui définit l'ensemble des propositions utilisées dans la description des préconditions des opérateurs de α, $\mathit{effects}^+(\alpha)$ et $\mathit{effects}^-(\alpha)$ qui représentent respectivement l'ensemble des propositions utilisées dans la description des effets négatifs et positifs des opérateurs de α. Nous supposons que ces trois ensembles constituent la partie publique d'un agent, i.e., les agents peuvent accéder à ces informations à n'importe quel moment du processus de planification. En d'autres termes, en rendant publiques ces informations, un agent publie les propriétés du monde qu'il est en mesure de modifier.

Définition 5.5 (Interaction négative)**.** *Une action* a *d'un graphe de planification d'un agent* α menace *l'activité d'un agent* β *ssi une proposition* $p \in \mathit{effects}^-(a)$ *est unifiable avec une proposition* q *telle que* $q \in \mathit{precond}(\beta)$ *ou* $q \in \mathit{effects}^+(\beta)$*.*

Définition 5.6 (Interaction positive)**.** *Une action* a *d'un graphe de planification d'un agent* α assiste *l'activité d'un agent* β *ssi une proposition* $p \in \mathit{effects}^+(a)$ *est unifiable avec une proposition* q *telle que* $q \in \mathit{precond}(\beta)$ *ou* $q \in \mathit{effects}^+(\beta)$*.*

Soulignons que ces deux définitions revêtent un caractère particulièrement important dans le cadre de notre modèle. C'est en effet, sur la base de ces interactions, que les agents seront capables de planifier en intégrant les activités des autres agents.

Définition 5.7 (Problème de planification multi-agent)**.** *Un* problème de planification multi-agent *est un couple* $(\mathcal{A}, g)$ *où* $\mathcal{A}$ *représente l'ensemble des agents devant résoudre le problème et* g *le but à atteindre, i.e., l'ensemble des propositions qui doivent être satisfaites par les agents après l'exécution du plan.*

Nous faisons ici l'hypothèse restrictive que l'union des croyances des agents d'un problème de planification est cohérente (cas classique de la planification mono-agent), i.e, pour deux agents α et $\beta \in \mathcal{A}$, si une proposition $p \in s_0^\alpha$ alors $\neg p \notin s_0^\beta$. Cependant, aucune hypothèse n'est faite sur le possible partage de croyances entre les agents en termes de faits ou d'opérateurs.

Exemple 5.1 (Dockers)**.** Considérons un exemple simple de problème multi-agent composé de trois agents dockers : deux dockers α et β capables de charger et de décharger des conteneurs à partir de camions et, un troisième γ capable de les déplacer. Le premier docker α est à l'emplacement *l1*. Le second docker β est à l'emplacement *l2*. Le but du problème est $g = \{$*(at c1 l2) (at c2 l1) (at t1 l2) (at t2 l1)*$\}$. Nous donnons ci-dessous la définition des opérateurs du problème ainsi que son état initial (cf. figure 5.1) :

```
;; Le docker charge un conteneur ?c dans le camion ?t en position ?l
(:action load
```

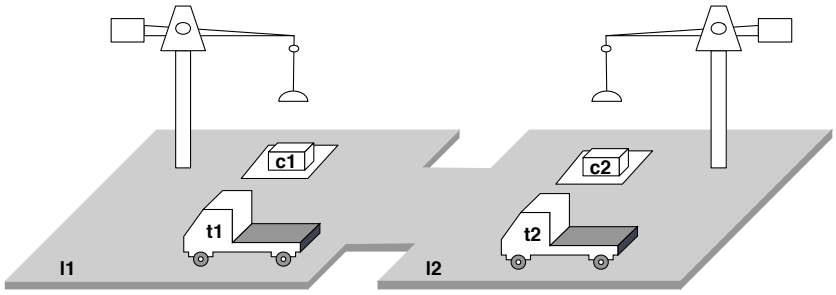


FIGURE 5.1 – État initial de l'exemple :

$$s_0^{\alpha} = \{(at\ c1,\ l1),\ (at\ t1\ l1)\}$$
$$s_0^{\beta} = \{(at\ c2\ l2),\ (at\ t2\ l2)\}$$
$$s_0^{\gamma} = \{(at\ t1\ l1),\ (at\ t2\ l2)\}$$

```
    (:parameters ?l - location ?c - container ?t truck)
    (:precondition (and (at ?l ?t) (at ?c ?l)))
    (:effect (and (not (at ?c ?l)) (in ?c ?t))))

;; Le docker décharge un conteneur ?c du camion ?t en position ?l
(:action unload
    (:parameters ?l - location ?c - container ?t truck)
    (:precondition (and (at ?l ?t) (in ?c ?t)))
    (:effect (and (not (in ?c ?t)) (at ?c ?l))))

;; Le docker déplace le camion ?c d'une position ?l1 à une position ?l2
(:action move
    (:parameters ?t truck ?l1 ?l2 - location)
    (:precondition (and (at ?t ?l1)))
    (:effect (and (not (at ?t ?l1)) (at ?t ?l2))))
```

Définition 5.8 (But individuel). *Le but individuel g_α d'un agent α pour un problème de planification multi-agent $(\mathcal{A}, g)$ avec $\alpha \in \mathcal{A}$ est défini par $g_\alpha = \{p \in g \mid p$ s'unifie avec une proposition $q \in effects^+(\alpha)\}$. Autrement dit, un agent ne peut accepter comme but que les propositions qui sont définies comme des effets des actions qu'il peut exécuter.*

Définition 5.9 (Plan). *Un* plan $\pi = \langle A_0, \ldots, A_n \rangle$ *est une séquence finie d'ensembles d'actions avec $n \geq 0$. Si $n = 0$, alors le plan π est le plan vide, noté $\langle\rangle$. Si les ensembles d'actions composant le plan sont des singletons, i.e., $A_0 = \{a_0\}, \ldots, A_n = \{a_n\}$, nous utiliserons la notation simplifiée $\pi = \langle a_0, \ldots, a_n \rangle$.*

Définition 5.10 (Plan individuel). *Un plan $\pi_\alpha = \langle A_0^\alpha, \ldots, A_n^\alpha \rangle$ est un* plan solution individuel *d'un problème de planification multi-agent $(\mathcal{A}, g)$ pour un agent $\alpha \in \mathcal{A}$ ssi tous les ensembles d'actions A_i^α sont indépendants et l'application des A_i^α à partir de s_0^α définit une séquence d'états $\langle s_0, \ldots, s_n \rangle$ telle que $g_\alpha \subseteq s_n$.*

Définition 5.11 (Plan multi-agent). *Un plan $\Pi = \langle A_0, \ldots, A_n \rangle$ est un* plan multi-agent *d'un problème de planification multi-agent $(\mathcal{A}, g)$ ssi il existe un sous-ensemble d'agents $\mathcal{A}'$ avec $\mathcal{A}' \subseteq \mathcal{A}$ tel que l'union des buts individuels g_α des agents $\alpha \in \mathcal{A}'$ est égale à g, que tous les agents $\alpha \in \mathcal{A}'$ possèdent un plan solution individuel $\pi_\alpha = \langle A_0^\alpha, \ldots, A_n^\alpha \rangle$ et que tous les ensembles d'actions $A_i = \bigcup A_i^\alpha$ sont indépendants.*

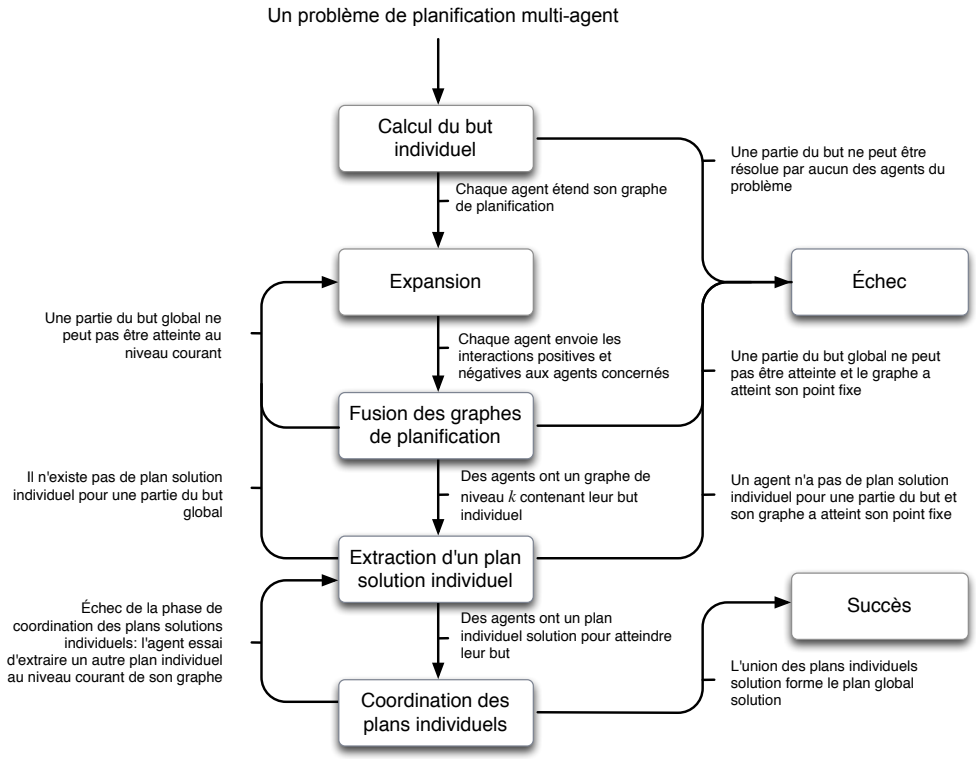


FIGURE 5.2 – Organisation de la procédure de planification distribuée par fusion incrémentale de graphes

5.2 Dynamique du modèle de planification

L'algorithme de planification distribuée par fusion incrémentale de graphes de planification (cf. figure 5.2) s'articule autour de cinq phases. Il réalise une recherche distribuée en profondeur d'abord par niveau, découvrant une nouvelle partie de l'espace de recherche à chaque itération. Dans un premier temps, chaque agent calcule les buts individuels associés aux agents du problème puis entre dans une boucle dans laquelle il étend itérativement son graphe de planification, fusionne les interactions négatives et positives provenant de l'activité des autres agents, et finalement effectue une recherche locale d'un plan individuel solution. Si les agents réussissent à extraire un plan solution individuel à partir de leur graphe de planification, ils tentent alors de les coordonner. Dans le cas contraire, chaque agent étend une nouvelle fois son graphe de planification. La boucle d'expansion, de fusion, et de recherche de plans individuels solutions se poursuit tant qu'aucun plan solution global n'est trouvé ou que la condition de terminaison sur échec que nous préciserons dans la suite de cette section n'est pas vérifiée.

5.2.1 Décomposition du but global

Tout d'abord, chaque agent calcule son but individuel (cf. définition 5.8), i.e., le sous-ensemble des propositions du but qui s'unifie avec un effet de ses opérateurs. Si une partie du but global n'apparaît dans aucun des buts individuels des agents, la phase de décomposition échoue, et la synthèse de plans se termine. Cet échec signifie qu'une partie du but global ne peut être atteinte, puisqu'aucun agent ne possède d'opérateurs capables de produire un sous-ensemble des propositions composant

le but global. Dans le cas contraire, les agents passent dans la phase d'expansion. Considérons l'exemple 5.1, les buts individuels des agents sont définis de la manière suivante : $g_\alpha = \{(at\ c2\ l1)\}$; $g_\beta = \{(at\ c1\ l2)\}$; $g_\gamma = \{(at\ t1\ l2), (at\ t2\ l1)\}$. Il est important de noter qu'un agent peut ne pas avoir de but individuel, mais être nécessaire à la synthèse d'un plan. Dans ce cas, l'agent étend malgré tout son graphe pour lui permettre d'évaluer les actions d'aide qu'il peut proposer aux autres agents.

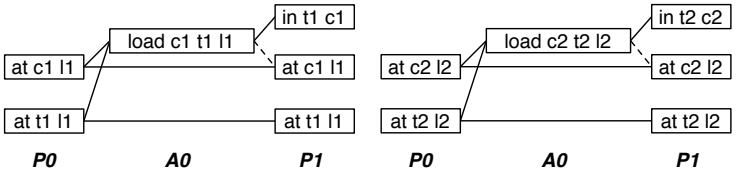
5.2.2 Expansion du graphe de planification

Cette phase débute par l'initialisation, par chaque agent, de son graphe de planification à partir de ses croyances. Puis, chaque agent étend son graphe en respectant la définition 5.2. La procédure d'expansion d'un graphe de planification G à un niveau i est donnée par l'algorithme 7. La procédure est identique à celle du planificateur Graphplan. Lorsque l'expansion de son graphe est terminée, l'agent envoie aux agents concernés les actions associées à son graphe qui peuvent interférer soit de manière positive (cf. définition 5.6) soit de manière négative (cf. définition 5.5). La figure 5.3 donne les graphes de planification associés aux agents α figure (a), β figure (b) et γ figure (c) après la première expansion de leur graphe : les boîtes au niveau P_i représentent des propositions et les boîtes au niveau A_i des actions ; pour simplifier, les relations d'exclusion mutuelles ne sont pas représentées ; les lignes pleines modélisent les préconditions et les effets positifs des actions tandis que les lignes pointillées modélisent les effets négatifs ; finalement, les boîtes en gras définissent les propositions du but lorsqu'elles sont atteintes sans mutex. Le graphe de γ contient son but individuel exempt d'exclusion mutuelle au niveau 1. En revanche, les graphes de planification de α et β ne contiennent pas leurs buts individuels respectifs.

Algorithme 7 : Expand(G, i)

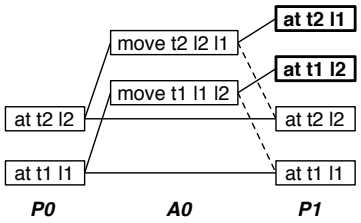
Let $G = \langle P_0, \mu P_0, A_0, \mu A_0, \ldots, A_{i-1}, \mu A_{i-1}, P_i, \mu P_i \rangle$ and i the level of G to expand
$A_i \leftarrow A_{i-1}$ *;; init ith action level*
$P_{i+1} \leftarrow P_i$ *;; init i+1th proposition level*
foreach $a \in A$ **do** *;; add applicable actions to level i*
  if $a \notin A_i$ *and* $\mathit{precond}(a) \subseteq P_i$ *and* $\mathit{precond}(a) \cap \mu P_i = \emptyset$ **then**
    $A_i \leftarrow A_i \cup \{a\}$
    $P_{i+1} \leftarrow P_{i+1} \cup \mathit{effects}(a)$
    $\forall p \in \mathit{precond}(a)$ add a *precondition-edge* from $p \in P_i$ to $a \in A_i$
    $\forall p \in \mathit{effects}^+(a)$ add an *add-edge* from $a \in A_i$ to $p \in P_{i+1}$
    $\forall p \in \mathit{effects}^-(a)$ add a *del-edge* from $a \in A_i$ to $p \in P_{i+1}$
;; update action mutual exclusions
$\mu A_i \leftarrow \{(a, b) \in A_i, a \neq b \mid a, b$ are dependent
  or $\exists (p, q) \in \mu P_{i-1} : p \in \mathit{precond}(a)$ and $q \in \mathit{precond}(b)\}$
;; update proposition mutual exclusions
$\mu P_{i+1} \leftarrow \{(p, q) \in P_{i+1}, p \neq q \mid \forall a, b \in A_i, a \neq b :$
  $p \in \mathit{effects}^+(a)$ and $q \in \mathit{effects}^+(b) \Rightarrow (a, b) \in \mu A_i\}$
return $\langle P_0, \mu P_0, A_0, \mu A_0, \ldots, P_i, \mu P_i, A_i, \mu A_i, P_{i+1}, \mu P_{i+1} \rangle$

Considérons maintenant les interactions négatives et positives entre les actions des différents graphes. Par exemple, l'action *(move t1 l1 l2)* de γ au niveau 0 de son



(a) Graphe de planification de l'agent α : but individuel non atteint

(b) Graphe de planification de l'agent β : but individuel non atteint



(c) Graphe de planification de l'agent γ : but individuel atteint sans mutex

FIGURE 5.3 – Graphe de planification des agents de l'exemple 5.1

graphe de planification menace l'activité de, α car elle supprime la précondition *(at t1 l1)* qui peut être nécessaire à l'application de l'opérateur *load* et *unload* de α. Par opposition, cette même action de γ assiste l'activité de β puisqu'elle produit l'effet *(at t1 l2)* qui peut être nécessaire au déclenchement des opérateurs *load* et *unload* de β. La liste complète des interactions entre les actions des agents au premier niveau de leur graphe est donnée par le tableau 5.1.

Actions	*Agent*	*Négative*	*Positive*
(load c1 t1 l1)	α	–	β
(load c2 t2 l2)	β	–	α
(move t1 l1 l2)	γ	α	β
(move t2 l2 l1)	γ	β	α

TABLE 5.1 – Table des interactions entre agents au niveau 0 de leur graphe de planification

5.2.3 Fusion des graphes de planification

Au cours de cette phase, chaque agent fusionne les actions provenant des autres agents à son graphe de planification pouvant interférer avec son activité. Cette étape est détaillée par l'algorithme 8. Puis chaque agent étend une nouvelle fois son graphe de planification pour ajouter les nouvelles actions qu'il peut dorénavant potentiellement réaliser avec l'aide des autres. À la fin de cette étape, le graphe de chaque agent contient désormais à la fois ses propres actions, mais également celles provenant des autres agents. Lors que l'étape de fusion est terminée, deux cas sont à envisager :

1. Un agent α nécessaire à la réalisation du but global possède un graphe qui a atteint son point fixe au niveau i et $g_\alpha \not\subseteq P_i$ ou $g_\alpha \in P_i$ et $g_\alpha \cap \mu P_i \neq$

Algorithme 8 : Merge(G, α, i, A)

Let $G = \langle P_0, \mu P_0, A_0, \mu A_0, \ldots, P_i, \mu P_i, A_i, \mu A_i, P_{i+1}, \mu P_{i+1} \rangle$ of the agent α
Let A be a set of threats or promotions actions to add at level i
foreach $a \in A$ **do** *;; add threats and promotions actions*
    $A_i \leftarrow A_i \cup \{a\}$
    precond $\leftarrow \{p \in \mathit{precond}(a) \mid p$ co-designates $q \in \mathit{precond}(\alpha)$ or $q \in \mathit{effects}(\alpha)\}$
    forall $p \in$ precond **do** *;; add preconditions*
        if $p \notin P_i$ **then**
            $j \leftarrow 0$
            $P_j \leftarrow P_j \cup \{p\}$
            while $j < i$ **do** *;; propagate new precondition in previous level*
                $P_{j+1} \leftarrow P_{j+1} \cup \{p\}$
                $A_j \leftarrow A_j \cup \{\textit{(no-op)}\}$
                add a *precondition-edge* from $p \in P_j$ to *(no-op)* $\in A_j$
                add an *add-edge* from *(no-op)* $\in A_j$ to $p \in P_{j+1}$
                $j \leftarrow j + 1$
        add a *precondition-edge* from $p \in P_i$ to $a \in A_i$
    effects $\leftarrow \{p \in \mathit{effects}(a) \mid p$ co-designates $q \in \mathit{precond}(\alpha)$ or $q \in \mathit{effects}(\alpha)\}$
    forall $p \in$ effects **do** *;; add positive and negative effects*
        if $p \notin P_{i+1}$ **then** $P_{i+1} \leftarrow P_{i+1} \cup \{p\}$
        if $p \in \mathit{effects}^+(a)$ **then** add an *add-edge* from $a \in A_i$ to $p \in P_{i+1}$
        else add a *del-edge* from $a \in A_i$ to $p \in P_{i+1}$
;; update mutual exclusions at level i
$\mu A_i \leftarrow \{(a, b) \in A_i, a \neq b \mid a, b$ are dependent or $\exists (p, q) \in \mu P_i$:
        $p \in \mathit{precond}(a)$ and $q \in \mathit{precond}(b)\}$
$\mu P_{i+1} \leftarrow \{(p, q) \in P_{i+1}, p \neq q \mid \forall a, b \in A_i, a \neq b$:
        $p \in \mathit{effects}^+(a)$ and $q \in \mathit{effects}^+(b) \Rightarrow (a, b) \in \mu A_i\}$
return $\langle P_0, \mu P_0, A_0, \mu A_0, \ldots, P_i, \mu P_i, A_i, \mu A_i, P_{i+1}, \mu P_{i+1} \rangle$

$\emptyset$: la synthèse du plan échoue et l'algorithme se termine. En effet, un sous-ensemble des propositions du but global ne peut plus être atteint.

2. Un agent α nécessaire à la réalisation du but global possède un graphe qui n'a pas atteint son point fixe au niveau, i mais $g_\alpha \not\subseteq P_i$ ou $g_\alpha \in P_i$ et $g_\alpha \cap \mu P_i \neq \emptyset$: tous les agents effectuent une nouvelle expansion de leur graphe de planification.

Finalement, si aucune des deux conditions n'est vérifiée, la phase de fusion se termine en garantissant qu'il existe un sous-ensemble des agents du problème capables d'atteindre le but global, et que chaque agent de ce sous-ensemble possède un graphe de planification de niveau i, tel que $g_\alpha \subseteq P_i$ et $g_\alpha \cup \mu P_i = \emptyset$.

À titre d'illustration, nous donnons à la figure 5.4 les graphes de planification des trois agents de l'exemple 5.1 après trois itérations d'expansion et de fusion. Afin de limiter l'ajout de propositions inutiles au niveau du graphe de planification pour ne pas pénaliser la recherche de plans, seules les propositions apparaissant dans la description des opérateurs d'un agent sont ajoutées. Par exemple, l'ajout de l'action *(move t2 l2 l1)* de l'agent γ dans le graphe de planification de l'agent α ne produira

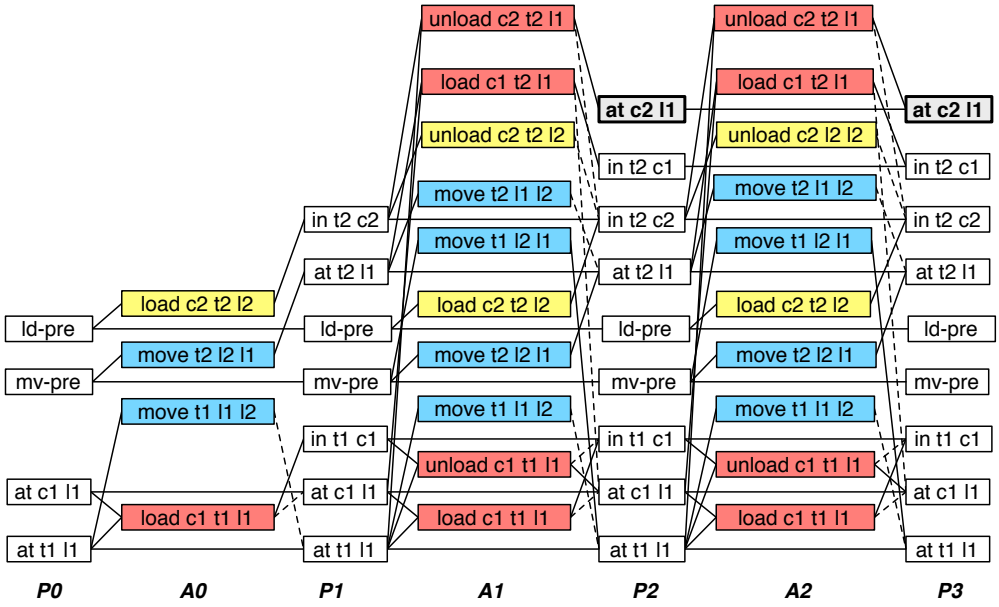
qu'un seul effet, *(at t2 t1 l1)*, au niveau 1 du graphe de α. Le même principe s'applique également aux préconditions des actions ajoutées. Toutefois, si l'action n'est pas liée par au moins une précondition au niveau propositionnel (c'est le cas de l'action *(move t2 l2 l1)*), nous ajoutons une précondition fictive pour garantir l'existence d'un chemin de P_0 à P_i, et ainsi permettre l'extraction d'un plan individuel solution.

5.2.4 Extraction d'un plan individuel

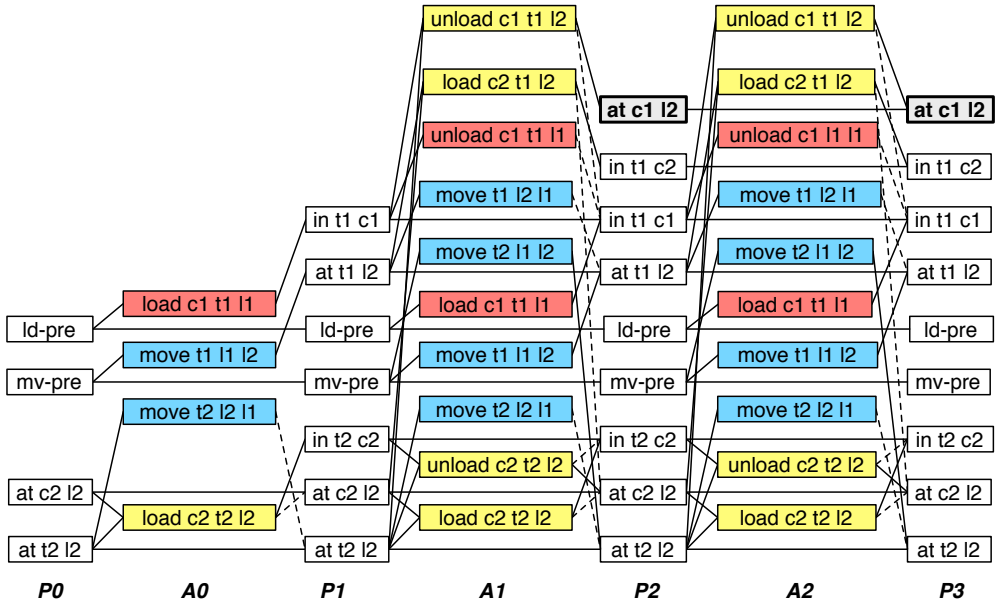
L'extraction d'un plan solution est effectuée en s'appuyant sur une technique de satisfaction de contraintes proposée par (KAMBHAMPATI, 2000). Cette technique présente deux principaux avantages : d'une part, elle améliore de manière significative les temps d'extraction des plans solutions et d'autre part, elle peut être facilement modifiée pour extraire un plan solution individuel intégrant les contraintes provenant des autres agents. L'extraction débute par l'encodage du graphe de planification sous la forme d'un CSP. Chaque proposition p à un niveau i du graphe de planification est assimilée à une variable CSP. L'ensemble des actions qui produisent p au niveau n constitue son domaine auquel on ajoute une valeur $\perp$ nécessaire à l'activation des actions dans le graphe. Les relations d'exclusion correspondent aux contraintes du CSP. Lorsque deux actions a_1 et a_2 sont mutuellement exclusives, alors pour toutes les paires de propositions p_1 et p_2 telles que a_1 peut produire p_1 et a_2 respectivement p_2, la contrainte $(p_1 = a_1) \Rightarrow (p_2 \neq a_2)$ est ajoutée au CSP. Lorsque deux propositions p_1 et p_2 sont mutuellement exclusives, on le traduit par une contrainte $\neg(p_1 \neq \perp) \wedge (p_2 \neq \perp)$. L'assignation des valeurs aux variables est dynamique, car chaque assignation, à un niveau donné, active des variables au niveau précédent en respectant la procédure classique d'extraction utilisée par le planificateur Graphplan. Initialement, seules les variables représentant le but individuel de l'agent sont actives. L'activation dynamique se traduit par des contraintes d'activation au niveau du CSP. Autrement dit, lorsqu' une variable, représentant une proposition, prend une certaine valeur, i.e., est produite par une action, alors d'autres variables, qui correspondent aux préconditions de l'action choisie, deviennent actives. L'extraction d'un plan solution individuel à partir du graphe de planification de l'agent est réalisée en résolvant le CSP. En partant des variables actives initialement, i.e., le but individuel de l'agent, la procédure d'extraction cherche à assigner une valeur ou une action, à chaque variable ou proposition, pour satisfaire l'ensemble des contraintes, i.e., des exclusions mutuelles. Si la résolution du CSP réussit, l'agent a extrait un plan solution individuel. Dans le cas contraire, l'agent n'est pas capable de produire un plan individuel solution. Deux cas doivent être considérés :

1. Un agent nécessaire à la réalisation du but global ne parvient pas à extraire un plan solution individuel même en poursuivant l'expansion de son graphe, l'algorithme se termine. Une partie du but global ne peut être atteinte.
2. Un agent nécessaire à la réalisation du but global ne parvient pas à extraire un plan solution individuel, mais peut espérer en trouver un en effectuant une nouvelle expansion de son graphe. Tous les agents effectuent une nouvelle expansion de leur graphe de planification.

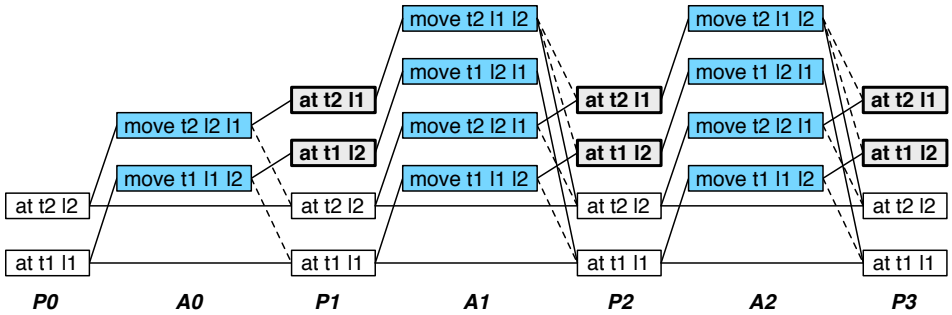
Finalement, la phase d'extraction se termine en garantissant qu'il existe un sous-ensemble des agents du problème capables d'atteindre le but global, et possédant un plan individuel solution.



(a) Graphe de planification de α.



(b) Graphe de planification de β.



(c) Graphe de planification de γ.

FIGURE 5.4 – Graphes de planification des agents de l'exemple des dockers après la phase d'expansion et de fusion au niveau 3. Les actions en jaune, en rouge et en bleu correspondent respectivement à celles des agents α, β et γ. Les propositions grisées représentent les propositions qui font partie du but individuel des agents.

5.2.5 Coordination des plans individuels

Il s'agit maintenant de vérifier la compatibilité des plans individuels dans le contexte multi-agent. Rappelons que lors de la phase de fusion, les agents ont ajouté dans leur graphe les actions provenant d'autres agents et pouvant interférer avec leur propre activité. Par conséquent, un plan individuel solution peut contenir des actions qui doivent être exécutées par d'autres agents. Autrement dit, un plan solution individuel est un plan conditionnel, i.e., un plan exécutable si certaines contraintes sont vérifiées. Dans notre cas, ces contraintes sont définies par des couples (a, i) ou a représentent une action et i le niveau où a doit être exécutée. Considérons l'exemple des dockers et leur graphe de planification (cf. figure 5.4). Les agents α et β peuvent extraire respectivement un plan solution individuel au niveau 2 :

$$\begin{aligned} \pi_\alpha &= \langle \textit{(load c2 t2 l2), (move t2 l2 l1), (unload c2 t2 l1)} \rangle \\ \pi_\beta &= \langle \textit{(load c1 t1 l1), (move t1 l1 l2), (unload c1 t1 l2)} \rangle \end{aligned}$$

Le plan π_α est valide si le plan individuel de β respecte la contrainte $(\textit{(load c2 t2 l2)}, 0)$ et celui de γ la contrainte $(\textit{(move t2 l2 l1)}, 1)$. De manière symétrique, π_β est valide si le plan individuel de α respecte $(\textit{(load c1 t1 l1)}, 0)$ et celui de γ la contrainte $(\textit{(move t1 l1 l2)}, 1)$. De son côté, l'agent γ peut potentiellement extraire plusieurs plans individuels solution dont aucun n'implique des actions provenant des autres agents :

$$\begin{aligned} \pi_\gamma &= \langle \textit{(move t2 l2 l1), (move t1 l1 l2), (no-op), (no-op)} \rangle \\ \pi'_\gamma &= \langle \textit{(move t2 l2 l1), (move t1 l1 l2), (no-op)} \rangle \\ \pi''_\gamma &= \ldots \end{aligned}$$

Les agents débutent la phase de coordination par l'échange des contraintes de leur plan individuel, et tentent ensuite d'intégrer les contraintes reçues à leur plan. L'intégration des contraintes repose sur le principe de moindre engagement. Tout d'abord, l'agent teste si les contraintes peuvent être ajoutées directement à son plan individuel solution. Ce test ne nécessite aucune replanification, puisqu'il suffit de vérifier que l'action à ajouter n'est pas mutuellement exclusive avec une action déjà présente dans le graphe de planification de l'agent au même niveau. Dans notre exemple, ce mécanisme de coordination sera utilisé par les agents α et β pour prendre respectivement en compte les contraintes $(\textit{(load c2 t2 l2)}, 0)$ et $(\textit{(load c1 t1 l1)}, 0)$. Si ce premier mécanisme échoue, l'agent tente alors d'extraire un nouveau plan individuel solution en incluant les contraintes. En d'autres termes, il propage les contraintes dans le graphe de contraintes représentant son graphe de planification, et effectue l'extraction d'un nouveau plan individuel solution. Ce mécanisme sera utilisé par l'agent γ pour intégrer les contraintes $(\textit{(move t1 l1 l2)}, 1)$ et $(\textit{(move t2 l2 l1)}, 1)$ provenant des agents α et β. En conclusion, les plans individuels solutions de l'exemple 5.1 obtenus après la phase de coordination sont les suivants :

$$\begin{aligned} \pi_\alpha &= \langle \textit{(load c1 t1 l1), (no-op), (unload c2 t2 l1)} \rangle \\ \pi_\beta &= \langle \textit{(load c2 t2 l2), (no-op), (unload c1 t1 l2)} \rangle \\ \pi_\gamma &= \langle \textit{(no-op),\{(move t1 l1 l2), (move t2 l2 l1)\}, (no-op)} \rangle \end{aligned}$$

Finalement, si ce second mécanisme de coordination échoue, les agents retournent dans la phase d'extraction de plans individuels. L'algorithme de fusion

incrémentale de plans peut alors se terminer (si aucun autre plan individuel ne peut être extrait) ou encore nécessiter une nouvelle expansion des graphes de planification des agents.

5.3 Évaluation théorique et empirique

Nous proposons dans cette section de donner quelques éléments sur la complexité du processus par fusion incrémentale de graphes ainsi qu'une évaluation expérimentale de l'approche.

5.3.1 Éléments de complexité

Élément sur la taille des graphes de planification

Proposition 5.1. *La taille du graphe de planification d'un agent de niveau k et le temps nécessaire à son extension et à sa fusion sont polynomiaux en fonction de la taille du problème de planification multi-agent. La taille du graphe de planification supplémentaire induite par le processus de fusion pour un agent dépend uniquement des interactions positives et négatives entre ses actions et celles des autres agents du problème.*

Proposition 5.2. *Le graphe de planification résultant de la fusion des graphes des agents d'un problème de planification à un niveau k est identique au graphe de planification qui serait produit par l'approche centralisée au même niveau.*

La preuve des propositions 5.2 et 5.1 nécessite de prendre soin de détails techniques, mais reste conceptuellement simple. Soit un problème de planification multi-agent $(\mathcal{A}, g)$ et un agent $\alpha \in \mathcal{A}$ avec O_α l'ensemble de ses opérateurs et s_0^α l'état initial de ses croyances. Soit a le nombre d'actions qui peuvent être instanciées par α à partir de O_α sachant s_0^α. Notons δ_a le nombre d'actions qui peuvent être ajoutées au graphe de α issues des interactions positives et négatives avec les autres agents impliqués dans la résolution du problème. De plus, notons p le nombre de propositions qui peuvent être déduites à partir des actions de α et δ_p les propositions supplémentaires provenant de l'ajout des actions traduisant les interactions positives et négatives entre α et les autres agents. Le graphe de planification de α contient au plus :

- $k(p + \delta_p)$ propositions ;
- $k(a + \delta_a + p + \delta_p)$ actions (en incluant les no-op) ;
- $k(p + \delta_p)^2$ propositions mutex et
- $k(a + \delta_a + p + \delta_p)^2$ actions mutex (en incluant les no-op).

Par ailleurs, a, p, δ_a et δ_p sont polynomiaux en fonction de la taille du problème. Commençons par borner a et p. Si l'on se place dans le cadre de la planification classique, le nombre des symboles de constantes est fini. Par conséquent, si c est le nombre de constantes manipulées par α, κ une borne supérieure sur le nombre de paramètres des opérateurs de α et $e = \max_{o \in O_\alpha}\{|\mathit{effects}^+(o)|\}$ alors, $a \leq |O_\alpha| \times c^\kappa$ et $p \leq |s_0^\alpha| + e \times |O_\alpha| \times c^\kappa$.

Essayons maintenant de borner le nombre δ_a d'actions pouvant être ajoutées au graphe de α pour traduire les interactions entre l'activité de α et des autres. Deux cas extrêmes doivent être envisagés :

Cas 1. Si aucune action n'interagit avec les actions de α, alors aucune action n'est ajoutée à son graphe. Dans ce cas, le but individuel de α est complètement indépendant des autres agents et $\delta_a = 0$.

Cas 2. Si toutes les actions des autres agents interagissent avec celles de α alors celles-ci doivent toutes être ajoutées au graphe de α et $\delta_a \leq \Sigma_{\alpha \in \mathcal{A}}\{|O_\alpha| \times c^\kappa\}$.

Finalement, intéressons-nous à δ_p. De manière similaire, deux cas extrêmes doivent être envisagés.

Cas 1. Si aucune action n'interagit avec les actions de α, alors aucune action n'est ajoutée au graphe de α et $\delta(p) = 0$.

Cas 2. Si toutes les actions des autres agents interagissent avec celles de α alors celles-ci doivent toutes être ajoutées au graphe de α et $\delta_p \leq |S| + e' \times |O| \times c^\kappa$ avec S l'union des états initiaux des agents du problème, O l'union des opérateurs des agents du problème et $e' = \max_{o \in O}\{|\mathit{effects}^+(o)|\}$.

Par conséquent, dans le pire des cas, i.e., lorsque toutes actions des agents interagissent, la taille du graphe de planification d'un agent a la taille du graphe qui serait construit si le problème était résolu de manière centralisée.

Élément sur la complexité de la phase d'extraction d'un plan solution

La complexité de la phase d'extraction d'un plan solution individuel est directement liée à la taille des graphes de planification des agents. Supposons qu'un agent doive rechercher un plan solution individuel dans un graphe de planification possédant k niveaux, chaque niveau contenant $a + \delta_a$ actions et $p + \delta_p$ propositions. Supposons que chaque action possède au plus n préconditions et e effets et que finalement chaque proposition à un niveau donné soit produite par au plus s actions. En respectant l'encodage proposé par (KAMBHAMPATI, 2000), le problème de la recherche d'un plan solution s'encode en un problème CSP avec ;

— $\mathcal{O}(k(p+\delta_p))$ variables CSP, une pour chaque proposition du graphe de planification ;
— $\mathcal{O}(k(a+\delta_a)^2)$ contraintes binaires pour encoder les mutex entre actions ;
— $\mathcal{O}(k(p+\delta_p)^2)$ contraintes binaires pour encoder les mutex entre propositions ;
— $\mathcal{O}(snk(p+\delta_p))$ contraintes binaires pour encoder l'activation des actions aux différents niveaux du graphe.

La complexité de la résolution d'un problème CSP est bien connue. Le résultat le plus pertinent pour notre approche est le suivant : un problème CSP peut être résolu par chaînage arrière avec apprentissage en temps polynomial en fonction de la profondeur d de l'arbre de recherche produit par une recherche en profondeur d'abord et exponentiel en fonction de la largeur w de l'arbre des contraintes[2] du problème CSP (DECHTER, 2003). Précisément, l'extraction d'un plan solution individuel a une complexité en temps de $\mathcal{O}((k(p+\delta_p)^2) \times 2s^w)$.

Ceci étant dit, regardons comment, le facteur dominant w est influencé par les interactions entre les agents impliqués dans le problème de planification. Il a également été montré (DECHTER, 2003) que $d \leq log(v) \times w$ où v est le nombre de variables du problème CSP. Par conséquent, w peut-être borné par :

$$k(p+\delta_p)/log(k(p+\delta_p)) \leq w \leq k(p+\delta_p)$$

Il est important de noter que :

2. Un graphe de contraintes est un graphe dans lequel chaque sommet représente une variable du problème CSP. Un arc relie deux sommets si une contrainte porte sur les deux variables sous-jacentes. L'absence d'arc entre deux sommets indique qu'il n'y a pas de contrainte directe entre les deux variables. De façon informelle, la largeur d'un graphe est une mesure qui traduit la façon dont les sommets sont liés.

1. w et $\delta(p)$ fournissent des mesures quantitatives du niveau d'interdépendance du problème de planification à résoudre ;
2. la complexité de l'extraction du plan solution individuel ne dépend pas du nombre d'agents, mais bien uniquement du niveau d'interdépendance du problème.

Élément sur la complétude et la correction

La procédure de planification distribuée par fusion incrémentale de graphes est donnée par l'algorithme 9.

Proposition 5.3. *Soit un problème de planification multi-agent $(\mathcal{A}, g)$, la procédure de planification par fusion incrémentale de graphes est correcte et complète. Elle retourne échec si et seulement si le problème n'a pas de solution ou un plan solution global permettant aux agents A d'atteindre le but g.*

Afin d'esquisser la preuve de cette proposition, reformulons le problème de planification. Supposons que tous les agents aient trouvé un plan solution individuel à un niveau k de leur graphe de planification. Chaque agent peut atteindre son but individuel s'il exécute les actions dont il a la responsabilité, mais également si les actions de son plan impliquant d'autres agents peuvent être exécutées par les agents concernés. C'est le rôle de la phase de coordination de vérifier cette dernière condition. Les actions planifiées, mais dont l'exécution incombe aux autres agents peuvent être vues comme des points de coordination dans le plan solution global. Autrement dit, les agents peuvent décider individuellement d'exécuter n'importe quelles actions avant ou après du moment que ces actions particulières sont exécutées au moment opportun et que leur but individuel reste atteignable. Le principe de l'étape de coordination de la procédure de planification par fusion incrémentale de graphes est d'énumérer pour chaque niveau tous ces points de coordination, garantissant ainsi que toutes les combinaisons de plans individuels solutions sont testées. Par ailleurs, la procédure de recherche distribuée en profondeur itérative qui repose sur l'expansion et la fusion successives niveau par niveau des graphes de planification garantit qu'aucun niveau ne sera oublié.

5.3.2 Expérimentations

Nous proposons ici une comparaison de l'approche de planification par fusion incrémentale de graphes (DPGM) avec deux travaux très proches : DisCSP (BRAFMAN et DOMSHLAK, 2008) et MA-A* (NISSIM et BRAFMAN, 2012). Pour plus de détails, le lecteur peut se référer à (DURKOTA et KOMENDA, 2013). L'implémentation de DPGM s'appuie sur le solveur CSP Minion [3].

Les expérimentations ont été réalisées sur cinq domaines : trois adaptés dans un contexte multi-agent et issus de la compétition internationale de planification (IPC) : Logistics, Rovers et Satellites et deux domaines développés spécifiquement pour l'évaluation de notre approche Linear-Logistics (un paquet doit être transporté de ville en ville par plusieurs agents formant une chaîne) et Deconfliction (des agents sur une grille doivent permuter leur position sans rentrer en collision). Les expérimentations ont été réalisées sur une machine équipée d'un processeur octo-core cadencé à 3,6GHz. Pour chaque expérimentation, la mémoire a été limitée à 2,5Gb et le temps maximum alloué est de 10 minutes. Les métriques utilisées sont le temps de résolution pour trouver un plan solution et le volume de

3. https://constraintmodelling.org

Algorithme 9 : DPGM($\mathcal{A}$, α, g)

$i \leftarrow 0$, $P_0 \leftarrow s_0$, $G \leftarrow \langle P_0 \rangle$, $\nabla \leftarrow \emptyset$
$g_\alpha \leftarrow \{p \in g \mid p$ co-designates $q \in \mathit{effects}^+(\alpha)\}$
if *the union of the individual goals of* $\mathcal{A}$ *differs from* g **then return failure**
state $\leftarrow$ EXPANSION
while state $\neq$ SUCCESS **do**
  switch state **do**
    case EXPANSION **do**
      $i \leftarrow i + 1$
      `Expand`(G, i)
      broadcast threats and promotions actions at the level i
      gather the threats and the promotions from the other agents in A
      state $\leftarrow$ MERGING
    case MERGING **do**
      `Merge`(G, α, i, A)
      if $g_\alpha \notin P_i$ *or* $g_\alpha \cap \mu P_i \neq \emptyset$ **then**
        if `Fixedpoint`(G) **then** broadcast $\langle$FAILURE$\rangle$
        else broadcast $\langle$EXPANSION$\rangle$
      else broadcast $\langle$EXTRACTION$\rangle$
      gather the state of the procedure from the other agents
      if *there is an agent in state* FAILURE **then return failure**
      if *there is an agent in state* EXPANSION **then** state $\leftarrow$ EXPANSION
      else state $\leftarrow$ EXTRACTION
    case EXTRACTION **do**
      $\pi_\alpha \leftarrow$ `Extract`$(G, g_\alpha, \emptyset)$
      if $\pi_\alpha =$ FAILURE *and* `Fixedpoint`(G) **then**
        if $\eta = |\nabla(i)|$ **then** broadcast $\langle$FAILURE$\rangle$
        else broadcast $\langle$EXPANSION$\rangle$, $\eta \leftarrow \nabla(i)$
      else broadcast $\langle$COORDINATION$\rangle$
      gather the state of the procedure from the other agents
      else if *there is an agent in state* FAILURE **then return failure**
      else if *there is an agent in state* EXPANSION **then** state $\leftarrow$ EXPANSION
      else state $\leftarrow$ COORDINATION
    case COORDINATION **do**
      broadcast $\langle$ `Constraints`(π_α) $\rangle$
      gather the coordination constraints from the other agents in C
      $\pi_\alpha \leftarrow$ `Extract`(G, g_α, C)
      if $\pi_\alpha =$ FAILURE **then** broadcast $\langle$EXTRACTION$\rangle$
      else broadcast $\langle$SUCCESS$\rangle$
    gather the state of the procedure from the other agents
    if *there is an agent in state* EXTRACTION **then** state $\leftarrow$ EXTRACTION
    else state $\leftarrow$ SUCCESS
return π_α

Domaines	DPGM	DisCSP	MA-A*
rovers-2-8-4	3,8s/69Kb	**1,4s/0,8Kb**	22,4s/52Kb
rovers-3-12-6	20,3s/234Kb	**7,9s/1,7Kb**	230,0s/2,5Mb
rovers-4-15-8	–	**62,3s/3,1Kb**	–
logistics-4-2-2-2	**0,3s/34Kb**	0,6s/15Kb	0,8s/77Kb
logistics-6-4-2-4	**0,6s/136Kb**	38,5s/7,1Mb	2,0s/320Kb
linear-logistics-6-7	**0,5s/167Kb**	–	1,7s/87Kb
linear-logistics-8-9	**0,7s/417Kb**	–	4,7s/254Kb
linear-logistics-10-11	**0,9s/859Kb**	–	15,4s/589Kb
linear-logistics-15-16	**1,6s/2,9Mb**	–	217,0s/4,2Mb
deconfliction-2	1,3/18kB	N/A	**0,9s/15Kb**
deconfliction-3	**0,2/13Kb**	N/A	1,2s/187Kb
deconfliction-4	–	N/A	3,8s/2.1Mb
satellite-6-12	**1,5s/266Kb**	4,4s/6,5Kb	7,4s/270Kb
satellite-8-16	**4,3s/793Kb**	–	37,5s/964Kb
satellite-10-20	**12,7s/1,8Mb**	–	189,0s/2,5Mb

TABLE 5.2 – Comparaison empirique de la planification par fusion incrémentale de graphes avec DisCSP et MA-A* en terme de temps et de volume de données échangées : N/A indique que le plan produit par le planificateur n'était pas correct et – indique que le temps ou la mémoire alloués ont été dépassés

données échangées entre agents. Les résultats sur quelques problèmes sont présentés de manière synthétique dans le tableau 5.2. Les chiffres entre tirets indiquent les caractéristiques des problèmes avec la sémantique suivante :

— le problème rovers-2-8-4 est un problème composé de 2 rovers, 8 points de passage et 4 objectifs, etc.
— le problème logistics-4-2-2-2 est composé de 4 agents (2 camions et 2 avions) situés dans 2 villes possédant 2 lieux d'expédition (1 aéroport et 1 entrepôt) et 2 paquets sont à déplacer, etc.
— le problème linear-logistics-6-7 est composé de 6 camions et 7 villes, etc.
— le problème defliction-2 est composé de 2 robots devant permuter leur position, etc.
— le problème satellite-6-12 est composé de 6 satellites et de 12 directions, etc.

Les résultats montrent que DPGM est globalement plus efficace que DisCSP et que MA-A* en temps, mais également en volume de données échangées entre agents. DPGM tire ses bons résultats de son mécanisme de factorisation de l'espace de recherche. Toutefois, on peut nuancer ces résultats en faisant deux observations :

1. DisCSP est plus performant lorsque la combinatoire des problèmes de planification est élevée au niveau de la phase de planification individuelle (c'est le cas des problèmes du domaine Rovers). Ceci s'explique par l'utilisation du planificateur FastForward (HOFFMANN et NEBEL, 2001) qui est particulièrement efficace sur les problèmes du domaine Rovers.
2. MA-A* est plus performant lors que le couplage des problèmes est élevé (c'est le cas des problèmes du domaine Deconfliction).

5.4 Conclusion

Nous avons proposé dans ce chapitre un modèle générique et compétitif pour la synthèse distribuée de plans par un groupe d'agents, appelé *planification distribuée par fusion incrémentale de graphes*. Le modèle est correct et complet, et unifie de manière élégante les différentes phases de la planification distribuée au sein d'un même processus. En outre, il permet aux agents de limiter les interactions négatives entre leurs plans individuels, mais également de prendre en compte leurs interactions positives, i.e., d'aide ou d'assistance, au plus tôt, i.e., avant l'extraction des plans individuels.

D'un point de vue technique, de nombreuses améliorations sont possibles pour augmenter les performances de notre approche. En particulier, il est possible d'utiliser un cache pour mémoriser les contraintes de coordination échangées entre les agents et d'utiliser ces informations pour élaguer des branches de l'espace de recherche. En effet, si une contrainte de coordination ne peut être intégrée dans le plan individuel d'un agent cela signifie que tout ensemble de contraintes contenant cette contrainte ne pourra être solution. Cette technique peut être vue comme une généralisation dans un contexte multi-agent du cache utilisé par le planificateur Graphplan pour accélérer la recherche d'un plan solution.

D'un point de vue scientifique, l'approche proposée ouvre la voie à des extensions de ce travail notamment pour :

— la prise en compte de la robustesse des plans. Par robustesse, nous entendons la capacité d'un plan à tolérer plus ou moins d'aléas d'exécution ou de croyances erronées sur l'état du monde. Supposons que nous soyons capables de construire un faisceau de plans solutions, i.e., un plan contenant différentes alternatives. La question qui se pose alors est la suivante : comment choisir les plans individuels les plus robustes minimisant ainsi le risque d'échec à l'exécution ?

— l'extraction de plans temporels flottants (MARIS et RÉGNIER, 2008), i.e., de plans dans lesquels les actions sont représentées par des intervalles pouvant être décalés dans le temps. Bien que la causalité soit suffisante pour un grand nombre d'applications, la prise en compte des aspects temporels est parfois nécessaire. La notion de plans flottants apparaît particulièrement intéressante dans un contexte multi-agent, car ils possèdent une grande flexibilité qui peut être mise à profit au cours de la phase de coordination et d'exécution.

Chapitre 6

Planification multi-agent hiérarchique et partiellement ordonnée

Le problème de la synthèse distribuée d'un plan d'actions pour résoudre de manière collaborative une tâche complexe est un challenge qui reste ouvert. Dans ce chapitre, nous présentons une seconde de nos contributions dans le cadre de la planification distribuée (PELLIER et FIORINO, 2005b, 2007, 2009) dans laquelle la génération d'un plan global partagé est vue comme un raisonnement collaboratif et argumentatif portant sur les actions à réaliser pour atteindre un but. L'idée centrale de ce travail consiste à tirer parti de deux techniques de planification utilisées en planification centralisée : POP (*Partial Order Planning*) (PENBERTHY et WELD, 1992), qui se prête bien à la distribution du processus de planification, car elle ne nécessite pas la représentation explicite d'état, et la planification hiérarchique aussi appelée HTN (*Hierachical Task Networks*) (NAU et al., 2003) pour son expressivité et ses algorithmes performants.

La suite du chapitre est organisée de la manière suivante : dans un premier temps, nous présentons une vue générale de notre contribution ainsi qu'un exemple introductif qui nous servira de fil rouge tout au long du chapitre ; dans un second temps, nous introduisons les concepts et les notations utilisés ; puis nous présentons notre contribution : le modèle de planification distribuée hiérarchique et partiellement ordonnée.

6.1 Présentation générale du modèle de planification

Le principe du modèle de planification distribuée hiérarchique et partiellement ordonnée est donné à la figure 6.1. Au niveau multi-agent, chaque agent peut raffiner, réfuter ou réparer le plan partagé en cours de construction en s'inspirant de la procédure POP. Si la réparation d'un plan précédemment réfuté réussit, il devient plus robuste, mais peut encore être réfuté. Si la réparation du plan échoue, les agents abandonnent le plan courant et tentent d'explorer un autre plan. Il est également possible que certains plans soient abandonnés tout simplement parce qu'aucun agent n'est capable de raffiner un sous objectif du plan partagé. Comme dans un système argumentatif où les agents échangent des propositions et des contre-propositions, le plan courant est considéré comme une solution lorsque les cycles de propositions/contre-propositions se terminent et qu'aucune objection ne peut plus être formulée.

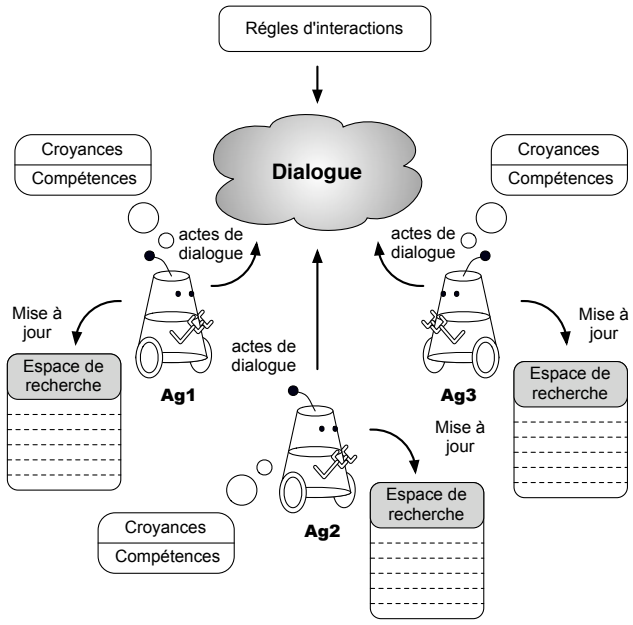


FIGURE 6.1 – Aperçu du modèle de planification distribuée hiérarchique partiellement ordonnée

Au niveau d'un agent, la spécificité de l'approche repose sur la capacité d'un agent à élaborer des plans à partir d'une base incomplète de croyances et de produire des plans qui peuvent être contradictoires avec leurs croyances. Autrement dit, un agent est capable de proposer des plans qui peuvent être exécutés sous réserve que certaines conditions soient vérifiées. Contrairement au modèle classique de planification, le processus de planification implanté dans nos agents n'échoue pas si une condition ou une contrainte n'est pas respectée. L'agent fait le pari qu'un autre agent pourra la vérifier ultérieurement pour lui. Les agents proposent donc des plans conditionnels ou plans sous hypothèses. Les conditions, i.e., les préconditions des actions non vérifiées, sont considérées comme des *buts ouverts* dans la terminologie POP. Bien évidemment, le but global n'est considéré comme atteint que lorsque tous les buts ouverts sont vérifiés ou qu'il existe un plan permettant de l'atteindre. L'objectif d'un agent est de produire des plans conditionnels en minimisant le nombre d'hypothèses puisque celles-ci deviennent autant de buts ouverts pour les autres agents. Illustrons le principe avec un exemple très simple. Supposons qu'une porte soit fermée : si un agent a besoin de pénétrer dans la pièce derrière cette porte et que la clé n'est pas dans le loquet, même s'il est capable d'atteindre tous ses objectifs derrière celle-ci, classiquement le processus de planification échoue. Une alternative consiste à supposer un instant que la clé est dans le loquet et à planifier les actions permettant d'atteindre les objectifs assignés. Dans ce cas, trouver la clé est un sous-but délégué aux autres agents. Pour permettre aux agents de produire de tels plans, nous avons développé un planificateur fondé sur une recherche de type HTN (*Hierarchical Task Network*) qui relaxe les contraintes d'application d'un opérateur dans un état.

Notre approche diffère des autres approches en trois points. Tout d'abord, contrairement aux approches qui considèrent le problème de planification distribuée comme un problème de contrôle et de coordination *a posteriori* des plans locaux des agents, e.g., par négociation (TOZICKA et al., 2014 ; ZLOTKIN et ROSENSCHEIN, 1990), par synchronisation (CARDOSO et BORDINI, 2016 ; LUIS et BORRAJO, 2014), etc., l'approche que nous proposons cherche à développer des mécanismes génériques afin

que des agents soient capables de construire conjointement et collaborativement un plan partagé pour atteindre un but commun. Deuxièmement, par élaboration nous entendons production d'un plan et non la simple instanciation d'un squelette de plan prédéfini (D'INVERNO et al., 2004). L'élaboration d'un plan repose sur la composition des compétences des agents et de leurs croyances. Finalement, par opposition aux approches de planification telles que (BRAFMAN et DOMSHLAK, 2008 ; NISSIM et BRAFMAN, 2012, 2014 ; PELLIER, 2010), notre approche est beaucoup plus expressive puisqu'elle repose sur une représentation hiérarchique des problèmes de planification et produit des plans partiels ordonnées. Produire des plans partiels ordonnés est très intéressant dans un contexte multi-agent : (1) un plan partiel peut capturer toute la complexité de la concurrence des activités d'un système multi-agent et (2) un plan partiel élaboré sur le principe de moindre engagement est très flexible. En effet, le choix final de l'ordre des actions peut être repoussé au moment de l'exécution en fonction du contexte observé, réduisant ainsi le besoin de replanifier.

Exemple introductif

Le scénario suivant permet d'illustrer ce que nous attendons par planification distribuée hiérarchique et partiellement ordonnée : Bob habite Grenoble et doit se rendre à New-York pour une conférence. Il décide d'organiser son voyage par internet en faisant appel à trois services web. Chaque service est représenté par un agent : un agent SNCF capable de réserver des billets de train ; un agent Airways offrant un service de réservation de billets d'avion et un agent Bank (représentant la banque de Bob) qui est en charge de payer les différentes réservations que Bob sera amené à réaliser. Imaginons maintenant le dialogue que les trois agents pourraient construire pour que Bob puisse se rendre à sa conférence :

Bob :	« Je suis à Grenoble et je dois me rendre à New-York. Pouvez-vous m'aider ? »
SNCF :	« Je ne peux malheureusement pas t'aider, je ne sais pas comment aller à New-York. »
Airways :	« En ce qui me concerne, je peux t'emmener à New-York à condition que tu sois capable de te rendre à Londres et que tu me paies la somme de 250 euros. »
SNCF :	« Je ne sais pas non plus comment aller à Londres. »
Bank :	« Je peux payer la somme de 250 euros, le compte de Bob est créditeur. »
Airways :	« Bon ce n'est pas grave, j'ai un autre vol en partance de Paris pour New-York à condition que Bob soit à Paris et que je reçoive la somme de 100 euros ».
Bank :	« Je peux payer les 100 euros du billet d'avion. »
SNCF :	« Il existe un train de Grenoble à Paris. En revanche, il faut que Bob puisse s'acquitter de la somme de 50 euros. »
Bank :	« Parfait, je crois que nous tenons la solution au problème de Bob, Je peux également payer les 50 euros du billet de train. »

Le plan solution est donc : « Prendre le train de Grenoble à Paris puis un vol de Paris à NY. » Sa construction ne repose pas sur une planification centralisée, mais sur la coopération de plusieurs agents planificateurs.

6.2 Définitions préliminaires

Dans cette section, nous définissons les notions préliminaires nécessaires à la formalisation de notre modèle de planification distribuée.

6.2.1 Les états de croyance

Un *état de croyance* est un ensemble de prédicats instanciés. Nous disons qu'un prédicat p est vérifié dans un état de croyance s si et seulement si p peut être unifié avec un prédicat de s tel que $\sigma(p) \in s$, où σ est la substitution résultant de l'unification de p avec le prédicat de s. Dans le cas contraire, nous considérons la propriété du monde représentée par p comme étant inconnue. Contrairement au chapitre précédent, nous ne faisons pas ici l'hypothèse du monde clos. Autrement dit, un prédicat p n'est pas vérifié dans un état s si et seulement si $\sigma(\neg p) \in s$.

6.2.2 Opérateurs et méthodes

Les opérateurs de planification sont définis comme des fonctions de transition au sens classique STRIPS (cf. définition 1.2) : actions instantanées, statiques, déterministes et observabilité totale.

Exemple 6.1. À titre d'exemple, nous donnons ci-dessous les opérateurs associés à l'agent Airways :

```
;; Airways déplace le passager ?p de la ville ?from à la ville ?to
(:action move
    (:parameters ?p - passager ?from ?to - location)
    (:precondition (and (flight ?from ?to) (at ?p ?from ))
    (:effect (and (not (at ?p ?from)) (at ?p ?to))))

;; Airways réserve le vol ?from ?to pour le passager ?p
(:action book
  (:parameters ?p - passager ?from ?to - location ?s - price)
    (:precondition (and (flight ?from ?to) (receive-cash ?p (price ?from ?
    to))
                    (>= (is-available ?from ?to) 1)))
    (:effect (and (decrease (is-available ?from ?to) 1))))
```

Nous complétons la définition des opérateurs en ajoutant le concept de *méthode* utilisé dans la planification hiérarchique (Nau et al., 2003). Contrairement à un opérateur qui décrit un ensemble d'actions, une méthode définit un ensemble de décompositions d'une tâche en actions pouvant être réalisées par un agent.

Définition 6.1 (Méthode). *Une* méthode m *est un triplet de la forme* $(name(m), precond(m), expansion(m))$ *où* $name(m)$ *est une expression de la forme* $n(x_1, \ldots, x_k)$ *telle que* n *représente le nom de la méthode et* $x_1, \ldots, x_k$ *ses paramètres,* $precond(m)$ *représente les préconditions (i.e., un ensemble de prédicats) devant être vérifiées dans l'état des croyances de l'agent pour que* m *soit appliquée et* $expansion(m)$ *définit la séquence d'opérateurs ou de méthodes à accomplir pour réaliser* m.

Exemple 6.2. Nous donnons ci-dessous la méthode de recherche d'un vol de l'agent Airways

```
;; Airways cherche un vol ?from ?to pour le passager ?p
(:method search-flight
    (:parameters ?p - passager ?from ?to - location)
    (:precondition (and (flight ?from ?to) (>= (is-available ?from ?to) 1)
    ))
    (:expansion (and (book ?p ?from ?to) (move ?p ?from ?to))))
```

6.2.3 Agent et Problème

Définition 6.2 (Agent)**.** *Un* agent α *est un couple* (O, M, s_0) *où* O *définit l'ensemble des opérateurs que peut réaliser* α*,* M *l'ensemble des méthodes que peut déclencher* α *et* s_0 *ses croyances initiales sur le monde.*

Il reste maintenant à définir un problème de planification. Un problème doit spécifier les états initiaux des croyances des agents, les opérateurs et méthodes qu'ils peuvent appliquer ainsi que le but qu'ils doivent réaliser. Le but est représenté par un ensemble de propositions décrivant les propriétés du monde qui doivent être vérifiées.

Définition 6.3 (Problème de planification)**.** *Un* problème de planification $\mathcal{P}$ *est un triplet* $(s_0, \mathcal{O}, g)$ *où* s_0 *et* $\mathcal{O}$ *représentent respectivement l'union des croyances, i.e., l'état initial du problème de planification, et les opérateurs (méthodes incluses) des agents et* g *définit un ensemble cohérent de propositions, i.e., les propriétés du monde devant être atteintes par les agents.*

Nous faisons l'hypothèse restrictive comme au chapitre précédent que l'union des croyances des agents d'un problème de planification est cohérente (cas classique de la planification mono-agent), i.e., pour deux agents α et β, si une proposition $p \in s_0^\alpha$ alors $\neg p \notin s_0^\beta$. Cependant, aucune hypothèse n'est faite sur le possible partage de croyances entre les agents en termes de faits ou d'opérateurs.

Exemple 6.3. La base de croyances des agents de notre exemple fil rouge est donnée ci-dessous dans l'ordre suivante : l'agent Airways α, l'agent bank β et l'agent SNCF σ. Le but des agents est défini formellement par $g = \{(at\ bob\ NewYork)\}$. Par souci de concision, nous donnons ici pour chaque agent l'état initial simplifié permettant de résoudre le problème.

$$s_0^\alpha = \left\{ \begin{array}{l} (flight\ London\ NewYork), \\ (flight\ Paris\ NewYork), \\ (flight\ NewYork\ Paris), \\ ((is\text{-}available\ London\ NewYork) = 8), \\ ((is\text{-}available\ Paris\ NewYork) = 10), \\ ((is\text{-}available\ NewYork\ Paris) = 2), \\ ((price\ London\ NewYork) = 250), \\ ((price\ Paris\ NewYork) = 100), \\ ((price\ NewYork\ Paris) = 100), \end{array} \right\}$$

$$s_0^\beta = \left\{ \begin{array}{l} ((account\ Bob) = 1000), \\ ((allowed\text{-}overdraft\ Bob) = 300) \end{array} \right\}$$

$$s_0^\sigma = \left\{ \begin{array}{l} \textit{(train Grenoble Paris)}, \\ \textit{(train Paris Grenoble)}, \\ \textit{((is-available Grenoble Paris) = 12)}, \\ \textit{((is-available Paris Grenoble) = 6)}, \\ \textit{((price Grenoble Paris) = 50)}, \\ \textit{((price Paris Grenoble) = 70)} \end{array} \right\}$$

6.2.4 Représentation des plans

Classiquement, un *plan* est un ensemble d'actions contenues dans une structure particulière exprimant des relations entre les actions. Un exemple de plan est donné à la figure 6.2. Dans le cas d'une séquence, la relation entre les actions est une relation d'ordre total. Le choix d'une telle structure semble trop restrictif pour s'appliquer dans un contexte multi-agent. En effet, elle ne permet pas de définir simplement la notion de plan mise en œuvre dans notre approche ni de décrire des actions concurrentes. Ceci nous amène à retenir pour notre approche la notion de plan partiel utilisée par les algorithmes de planification dans un espace de plans tels que (PENBERTHY et WELD, 1992). Pour illustrer tous les aspects d'un plan partiel, reprenons l'exemple 6.3 comme fil conducteur. Nous supposons qu'il existe un plan partiel initial constitué de deux actions proposées par l'agent α qui permet d'atteindre le but *(at Bob NewYork)* :

1. *(book Bob Paris NewYork)*
2. *(move Bob Paris NewYork)*

Regardons comment le plan partiel doit être raffiné par ajouts successifs d'actions [1] et de quelle manière s'effectue sa mise à jour. Cela nous permettra d'introduire de manière informelle la notion d'hypothèse ainsi que les quatre constituants d'un plan partiel : un ensemble d'actions, un ensemble de contraintes d'ordre, un ensemble de contraintes d'instanciation et un ensemble de liens causaux.

Les actions. Pour l'instant rien ne garantit au sein du plan partiel que Bob soit à Paris pour prendre son avion jusqu'à New-York. Par conséquent, la propriété *(at Bob Paris)*, requise par les préconditions de l'action *fly*, est une hypothèse formulée par le plan partiel initial. Pour vérifier cette hypothèse, l'agent σ propose de raffiner ce plan partiel en ajoutant la séquence suivante de deux actions :

1. *(book Bob Grenoble Paris)*
2. *(move Bob Grenoble Paris)*

De la même manière, rien ne garantit à l'agent α que le compte de Bob soit suffisamment approvisionné pour effectuer la réservation du billet d'avion Paris – New-York. Par conséquent, la précondition de l'action *(book Bob Paris NewYork)*, *(receive-cash Bob 100)* est également une hypothèse formulée par le plan partiel. Pour vérifier cette hypothèse, l'agent β propose d'ajouter l'action suivante :

1. *(pay bob 100)*.

1. Par abus de langage, nous utiliserons par la suite le terme générique d'action pour caractériser à la fois une action en tant qu'instance d'un opérateur de transformation et l'opérateur lui-même.

Les contraintes d'ordre. L'action proposée par l'agent σ *(move Bob Grenoble Paris)*, et celle proposée par l'agent β *(pay Bob 100)*, doivent être exécutées respectivement avant l'action *(move Bob Paris NewYork)* et *(book Bob Paris NewYork)* pour satisfaire les hypothèses formulées par le plan partiel initial. En effet, rien n'indique pour l'instant l'ordre dans lequel ces actions doivent être exécutées. Par conséquent, il est nécessaire d'ajouter des contraintes d'ordre précisant que *(move Bob Grenoble Paris)* doit être réalisée avant *(move Bob Paris NewYork)* et *(pay Bob 100)* avant *(book Bob Paris NewYork)*.

En revanche, est-ce que l'action *(move Bob Grenoble Paris)* doit être exécutée avant ou après *(pay Bob 100)* ? Les deux options sont possibles. Dans l'état actuel du plan partiel, rien n'oblige à trancher pour l'une ou l'autre des solutions. Nous appliquons ici le principe de *moindre engagement*. L'ajout d'une contrainte n'a lieu que si elle est strictement nécessaire. Si aucune autre contrainte d'ordre n'est ajoutée au plan partiel au cours du processus de planification, alors les actions proposées par l'agent σ et l'agent β pourront être exécutées de manière concurrente.

Les liens causaux. Pour le moment, nous savons ajouter des actions et des contraintes d'ordre à un plan partiel. Mais est-ce suffisant ? À cause de la représentation non explicite de la notion d'état courant (car distribué sur l'ensemble des agents), les contraintes d'ordre ne suffisent pas à garantir, par exemple, que Bob restera à Paris jusqu'à ce que l'action *(move Bob Paris NewYork)* soit réalisée. En effet, au cours du processus de planification, les agents peuvent trouver d'autres raisons de déplacer Bob dans une autre ville pour une correspondance et oublier la raison qui les a fait le déplacer à Paris. Par conséquent, il est nécessaire de coder explicitement au sein du plan partiel les raisons qui ont fait que les actions ont été ajoutées. Ainsi, dans notre exemple, il faut spécifier que l'action *(move Bob Grenoble Paris)* de l'agent σ a été ajoutée pour satisfaire la précondition *(at Bob Paris)* de l'action *(move Bob Paris NewYork)*.

La relation entre les actions *(move Bob Grenoble Paris)* et *(move Bob Paris NewYork)* portant sur la propriété *(at Bob Paris)* est appelée un *lien causal*. L'action *(move Bob Paris NewYork)* est appelée le consommateur et l'action *(move Bob Grenoble Paris)* le producteur. Autrement dit, un lien causal exprime qu'une propriété du monde nécessaire à l'exécution d'une action est satisfaite par les effets d'une autre action. En l'absence de lien causal, la précondition de l'action n'est pas vérifiée et sera considérée comme une hypothèse formulée par le plan partiel. Les hypothèses sont alors assimilées à des *sous-buts* devant être réalisés par les autres agents.

Notons qu'une action qui supporte une hypothèse doit toujours être réalisée avant l'action qui la formule. Par conséquent, un lien causal est toujours associé à une relation d'ordre, mais il est possible d'avoir une contrainte d'ordre sans lien causal. Toutefois, d'autres actions peuvent être intercalées entre les deux actions liées par un lien causal. Un lien causal n'est donc pas garant de l'absence de conflit entre deux actions.

Les contraintes d'instanciation. Il est nécessaire également de préciser les contraintes d'instanciation relatives aux variables manipulées par les opérateurs de transformation décrivant les actions. En effet, chaque opérateur, comme présenté dans la section 6.2.2 décrit un ensemble d'actions. L'unification des préconditions d'un opérateur avec l'état de croyance d'un agent peut définir plusieurs actions applicables à partir d'un même état. Il faut donc garantir, par exemple, que le nouvel opérateur *(move Bob Grenoble Paris)* concerne bien Bob ainsi que le même lieu

d'arrivée que le lieu de départ de l'opérateur *(move Bob Paris NewYork)*. Finalement, notons que certaines variables peuvent ne pas être instanciées. Les variables non instanciées traduisent le fait qu'un agent ne connaît pas, pour l'instant, la valeur exacte qui lui sera associée. Nous parlons alors d'action partiellement instanciée.

Pour résumer, nous avons ajouté au plan partiel des actions, des contraintes d'ordre, des liens causaux ainsi que des contraintes d'instanciation. Ces éléments constituent les éléments nécessaires à la formalisation de la notion de plan partiel utilisé dans notre approche.

Définition 6.4 (Plan partiel)**.** *Un* plan partiel *est un tuple* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ *dont les éléments sont les suivants :*

- $\mathcal{A} = \{a_0, \ldots, a_n\}$ *est un ensemble d'actions ;*
- $\prec$ *est un ensemble de contraintes d'ordre sur les actions* $\mathcal{A}$ *de la forme* $a_i \prec a_j$*, i.e.,* a_i *précède* a_j *;*
- $\mathcal{I}$ *est un ensemble de contraintes d'instanciation portant sur les variables des actions* $\mathcal{A}$ *de la forme* $?x = ?y$*,* $?x \neq ?y$*, ou* $?x = c$ *tel que* $c \in D_{?x}$ *et* $D_{?x}$ *est le domaine de* $?x$ *;*
- $\mathcal{C}$ *est un ensemble de liens causaux de la forme* $a_i \xrightarrow{p} a_j$ *tels que* a_i *et* a_j *sont deux actions de* $\mathcal{A}$*, la contrainte d'ordre* $a_i \prec a_j$ *existe dans* $\prec$*, la propriété* p *est un effet de* a_i *et une précondition de* a_j *et finalement les contraintes d'instanciation qui lient les variables de* a_i *et de* a_j *portant sur la propriété* p *sont contenues dans* $\mathcal{I}$*.*

Les hypothèses formulées par un plan partiel sont représentées par les préconditions des actions qui ne sont pas supportées par un lien causal.

Définition 6.5 (Hypothèse)**.** *Soit un plan partiel* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$*. Une hypothèse formulée par* π *est définie comme une précondition* p *d'une action* $a_j \in \mathcal{A}$ *telle que pour toutes actions* $a_i \in \mathcal{A}$*, le lien causal* $a_i \xrightarrow{p} a_j \notin \mathcal{C}$*. Nous notons respectivement* $open(\pi)$ *et* $open(a_j)$ *l'ensemble des hypothèses formulées par* π *et par* a_j*.*

De plus, l'ordonnancement partiel des actions implique qu'un plan partiel définit un ensemble de séquences d'actions totalement ordonnées respectant $\prec$.

Définition 6.6 (Linéarisation)**.** *Soit un plan partiel* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$*. On appelle linéarisation de* π *toute séquence d'actions* $\lambda = (\mathcal{A}, <, \mathcal{I}, \mathcal{C})$*, où* $<$ *est un ordre total sur* $\mathcal{A}$ *compatible avec* $\prec$*, qui définit une séquence de* $n+1$ *états* $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ *pour* $0 \leq i \leq n$ *avec*

$$s_i = (((s_{i-1} \cup open(a_{i-1})) - effects^-(a_{i-1})) \cup effects^+(a_{i-1})$$

Définition 6.7 (Complétion)**.** *Soit un plan partiel* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$*. On appelle complétion de* π *l'ensemble des linéarisations de* π*, noté* $completion(\pi)$*.*

Nous dirons que l'ensemble des contraintes d'ordre $\prec$ d'un plan partiel π est *cohérent* si $completion(\pi)$ est non vide. Cela signifie qu'il existe au moins une linéarisation possible de π. Pour tester la cohérence des contraintes d'ordre $\prec$ d'un plan partiel π, il faut vérifier que $\prec$ n'exprime aucun cycle de dépendance entre les actions de π. La vérification de cette propriété s'effectue en calculant la fermeture transitive de la relation d'ordre définie par $\prec$. Le calcul de la fermeture transitive permet de déterminer pour chaque couple d'actions a_i et a_j s'il existe une relation d'ordre[2].

2. Une relation d'ordre sur un ensemble E est une relation binaire dans E, à la fois réflexive, antisymétrique et transitive. Cette relation d'ordre est totale si deux éléments quelconques de E sont comparables sinon elle est partielle.

Finalement, l'absence de la notion d'état oblige à représenter les buts par une action particulière. Étant donné que les préconditions d'une action définissent les hypothèses potentielles pouvant être formulées par une action, les buts g sont représentés par une action fictive a_∞ qui ne possède pas d'effets. De manière similaire, la représentation de l'état initial nécessite l'introduction d'une action fictive a_0. Cette action ne possède pas de précondition, mais des effets qui représentent l'état initial. Notons que l'état initial global n'est pas accessible directement puisqu'il est réparti sur l'ensemble des agents. Par conséquent, cet état est construit au cours du processus de synthèse de plans par ajout d'effets à a_0.

6.2.5 Plans solutions et réfutations

Classiquement, un *plan-solution* se définit comme un chemin dans un espace d'états. Le passage d'un état à l'autre s'effectue par l'application d'une action, i.e., un opérateur complètement instancié, respectant la définition 6.6. Par conséquent, un plan solution pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ est une séquence d'actions décrivant un chemin d'un état initial s_0, représentant l'union des croyances des agents, à un état final s_n tel que $g \subseteq s_n$. Or, dans notre approche, nous devons tenir compte du fait qu'un plan partiel peut contenir des hypothèses et définit non pas une séquence, mais un ensemble de linéarisations. Par conséquent, toutes les linéarisations d'un plan partiel doivent décrire un chemin de l'état s_0 à s_n pour que le plan partiel soit un plan solution valide. En outre, il est clair que si un plan partiel π ne définit pas un ensemble de contraintes d'ordre $\prec$ cohérent, alors π ne peut être un plan solution. Ceci fournira un moyen d'éliminer des voies de recherche inutiles, en interdisant aux agents d'introduire des cycles de dépendances. Rappelons également que les contraintes d'instanciation utilisées dans notre modèle sont de trois types : les contraintes unaires de la forme $?x = c$, $c \in D_{?x}$ et les contraintes binaires de la forme $?x = ?y$ et $?x \neq ?y$. Il faut donc également ne garantir qu'aucune des contraintes d'instanciation de $\mathcal{I}$ n'exprime de contradiction, par exemple :

$$\mathcal{I} = \{?x = c_1, ?x = ?y, ?y = c_2, ?z \neq ?x, ?z = c_1\}$$

En conclusion, nous donnons la définition d'un plan solution :

Définition 6.8 (Plan-solution)**.** *Un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un plan-solution pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ si l'ensemble des contraintes d'ordre $\prec$ et l'ensemble des contraintes d'instanciation $\mathcal{I}$ sont cohérents et toutes les linéarisations $\lambda \in$ completion(π) définissent une séquence d'états cohérents $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ pour $0 \leq i \leq n$ tels que*

- *le but g est vérifié dans l'état s_n, i.e., $g \subseteq s_n$;*
- *λ ne formule aucune hypothèse, i.e., open$(\lambda) = \emptyset$.*

Malheureusement, la seconde partie de la définition, qui consiste à tester systématiquement pour chaque linéarisation d'un plan partiel si elle décrit une séquence d'états cohérents conduisant à un état but, ne définit pas une condition aisément calculable. Par conséquent, nous avons besoin de spécifier un ensemble de propriétés traduisant de façon pratique cette condition. Pour cela, nous la réexprimons en termes de *réfutations*. Bien qu'un plan partiel ne formule plus d'hypothèse, il peut ne pas être assez contraint pour garantir que toutes les séquences d'actions possibles définies par $\prec$ soient exemptes de conflit. En effet, un lien causal $a_i \xrightarrow{p} a_j$ n'interdit pas que d'autres actions soient exécutées entre a_i et a_j. Pour s'en persuader, considérons le plan partiel de la figure 6.3. Supposons que l'effet $\neg q$ soit

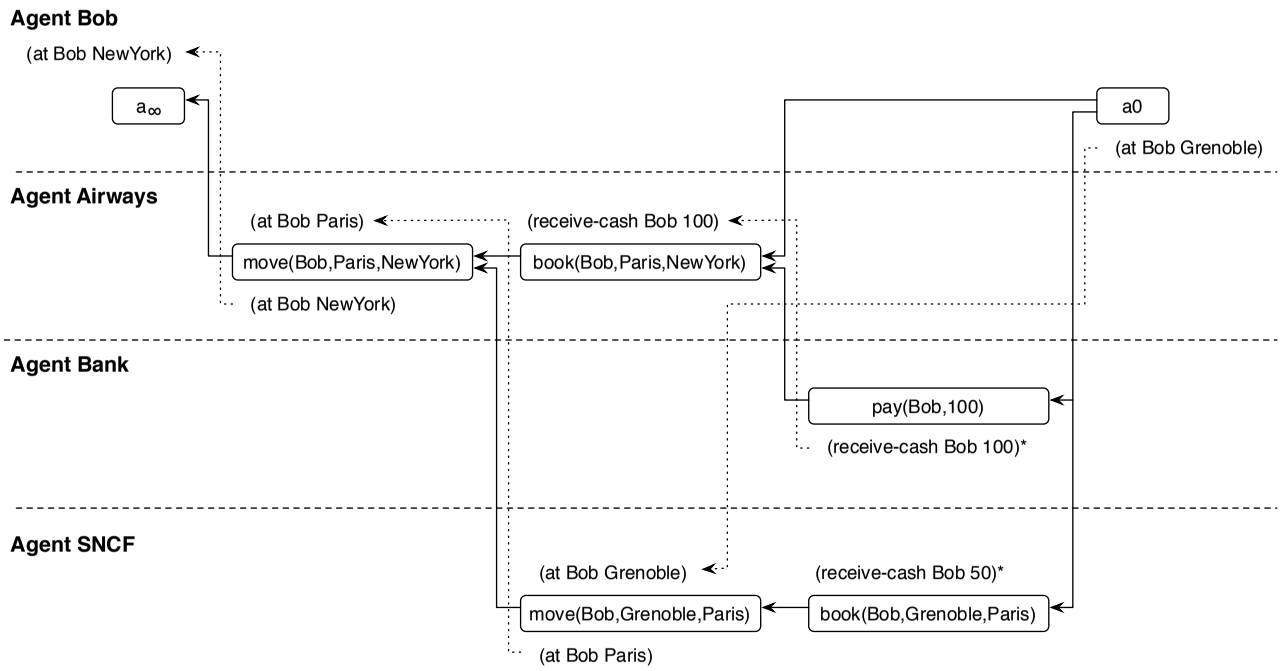


FIGURE 6.2 – Plan partiel initial de l'exemple 6.3 : les boîtes représentent les actions, les flèches pleines les contraintes d'ordre et les flèches pointillées les liens causaux entre les actions. Les contraintes d'instanciation sont décrites explicitement dans les actions et les propositions. Les propositions marquées d'un astérisque sont des hypothèses

produit par l'action a_k, et que q soit unifiable avec p. L'action a_k invalide potentiellement une précondition nécessaire à l'exécution de a_j. En l'absence de contrainte d'ordre entre a_k et les actions a_i et a_j, le plan partiel définit au moins une sous-séquence d'actions $\langle a_i, \ldots, a_k, \ldots, a_j \rangle$ invalide : la propriété du monde représentée par p n'est pas vérifiée dans l'état précédant l'exécution de a_j. Pour capturer cette condition et ainsi supprimer les séquences d'actions non valides, nous définissons ce que nous appelons une *réfutation*.

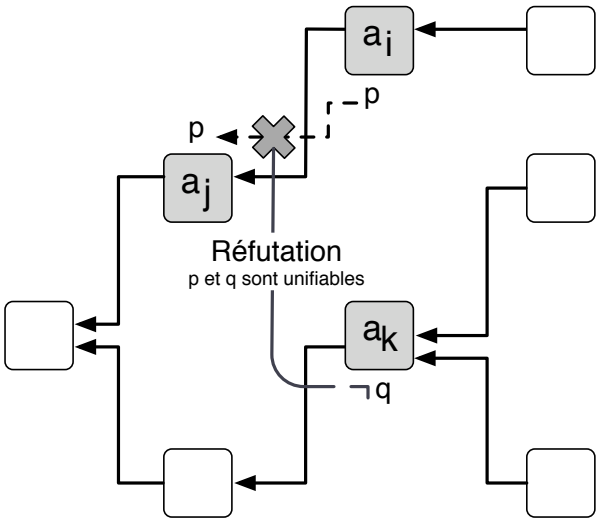


FIGURE 6.3 – Exemple de réfutation. p et q sont deux prédicats unifiables

Définition 6.9 (Réfutation). *Une réfutation portant sur un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un couple $(a_k, a_i \xrightarrow{p} a_j)$ tel que : (i) a_k a pour effet $\neg q$ avec p et q unifiables ; (ii) les contraintes d'ordre $a_i \prec a_k$ et $a_k \prec a_j$ sont cohérentes avec $\prec$ et (iii) les contraintes d'instanciation résultant de l'unification de p et q sont cohérentes avec $\mathcal{I}$.*

Par la suite, nous utiliserons le terme de *menace* pour caractériser l'ensemble des hypothèses et des réfutations d'un plan partiel.

Proposition 6.1. *Un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un plan solution pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ si les ensembles de contraintes d'ordre $\prec$ et d'instanciation $\mathcal{I}$ sont cohérents et π ne contient aucune menace.*

Le lemme suivant permet de prouver la proposition 6.1 :

Lemme 6.1. *Soit $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ un plan partiel et un lien causal $(a_i \xrightarrow{p} a_n) \in \mathcal{C}$. Nécessairement $p \in s_n$ s'il n'existe pas de réfutation $(a_k, a_i \xrightarrow{p} a_n)$.*

Preuve 6.1. *Preuve par induction sur la longueur de $\lambda \in completion(\pi)$:*

Cas de base : soit $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ avec $\mathcal{A} = \{a_0, a_\infty\}$. $completion(\pi) = \{\lambda\}$ et $\lambda = \langle a_0, a_\infty \rangle$. $s_0 = s_n$ et, par définition, il n'existe pas de réfutation possible ($\forall p \in s_0, p \in s_n$).

Induction : Supposons que le lemme est vérifié pour π ayant n actions. Montrons qu'il est également vrai pour π composée de $n+1$ actions. Soit $\lambda \in completion(\pi)$ avec $\lambda = \langle a_0, \ldots, a_{n-1}, a_n \rangle$ ($a_n = a_\infty$) et $\lambda' = \langle a_0, \ldots, a_{n-1} \rangle$. D'après l'hypothèse d'induction, $\forall (a_i \xrightarrow{p} a_{n-1})$ pour $0 \leq i < n-1$, $p \in s_{n-1}$ s'il n'existe pas de

réfutation $(a_k, a_i \xrightarrow{p} a_{n-1})$ *pour* $0 \leq k < n-1$. *Par définition,* $s_n = ((s_{n-1} \cup open(a_{n-1}) - effects^-(a_{n-1})) \cup effects^+(a_{n-1})$. *Par conséquent,* $\forall p \in s_n$, *soit* $p \in effects^+(a_{n-1})$, *soit* $p \in s_{n-1} \cup open(a_{n-1})$ *et* $p \notin effects^-(a_{n-1})$. *Dans le premier cas,* p *est produit par* a_{n-1} *et il n'y a pas de réfutation possible. Dans le second cas,* p *a été produit par* λ' *et n'est pas réfutée par* a_{n-1}. *Donc, dans tous les cas, le lemme est vérifié.*

Preuve 6.2. *Soit* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$. $\prec$ *et* $\mathcal{I}$ *sont cohérents et il n'y a pas de menace dans* π. *Donc,* $\forall \lambda \in completion(\pi)$, λ *définit une séquence d'états* $\langle s_0, \ldots, s_n \rangle$ *telle que* $s_i = (s_{i-1} - effects^-(a_{i-1})) \cup effects^+(a_{i-1})$ *car* $open(\pi) = \emptyset$. *Comme il n'y a pas de réfutation dans* π, $g \subseteq s_n$. *Cela se démontre par l'absurde. Supposons qu'il existe* $p \in g$ *et* $p \notin s_n$. *Comme* p *ne peut être une hypothèse* $(\exists (a_i \xrightarrow{p} a_n))$, *l'absence de* p *dans* s_n *est due à une réfutation d'après le lemme 6.1. Ceci est contradictoire avec l'absence de menace. Par conséquent,* $g \subseteq s_n$ *et* π *est un plan solution.*

6.3 Dynamique du modèle de planification distribuée

Nous formalisons dans cette section la dynamique du modèle de composition permettant à un groupe d'agents de raisonner conjointement à l'élaboration d'un plan solution. D'une part, les agents doivent être capables d'interagir en respectant un certain nombre de règles qui garantissent le bien-fondé du raisonnement produit et, d'autre part, ils doivent être capables de démontrer la validité des hypothèses formulées par les autres agents, de réfuter les plans partiels incorrects ou encore de les réparer lorsque ceux-ci ont été précédemment réfutés. Par conséquent, il faut distinguer deux types de mécanismes : les mécanismes qui spécifient quand un agent peut interagir avec un autre agent, e.g., pour réfuter un plan partiel, et les mécanismes qui servent de support à la production du contenu de l'interaction.

6.3.1 Principe

Chaque agent possède un tableau (JAGANNATHAN, DODHIAWALA et BAUM, 1989) dans lequel il enregistre les propositions des autres agents. D'un point de vue algorithmique, le tableau peut être vu comme un graphe orienté dont les nœuds représentent des plans partiels. Chaque arête sortant d'un nœud π est un opérateur qui transforme un plan partiel π en un plan partiel successeur π', traduisant ainsi les modifications proposées par les agents en termes de raffinement, de réfutation et de réparation. Par conséquent, le tableau définit l'état des interactions entre agents, mais également l'espace de recherche co-construit par les agents.

Les différents actes de dialogue utilisés dans notre approche sont donnés par le tableau 6.2. Ils se regroupent en deux niveaux : *les actes de niveau informationnel* qui permettent aux agents d'échanger des informations sur les plans partiels contenus dans leurs tableaux et *les actes de niveau contextualisation* qui permettent de modifier le contexte de l'interaction. C'est par l'intermédiaire de ces actes que les agents vont pouvoir débuter ou encore suspendre l'élaboration d'un plan solution.

Classiquement, le tableau de chaque agent est initialisé avec un plan partiel π_0 défini par :

— deux actions a_0, a_∞ telles que les préconditions de a_∞ représentent le but g soumis à l'ensemble des agents et les effets de a_0 l'état initial des croyances de l'agent ;

— une relation d'ordre $(a_0 \prec a_\infty)$;

Niveau	Actes
Informationnel	refine, refute, repair, failure
Contextualisation	prop.solve, prop.failure, prop.success, ack.failure, ack.success

TABLE 6.2 – Tableau des actes de dialogue

— les contraintes d'instanciation relatives à la description des préconditions et des effets de a_0, a_∞ ;
— un ensemble de liens causaux vide.

π_0 n'est pas un plan solution, car le but n'est soutenu par aucune relation causale. Les agents vont donc raffiner ce plan par l'ajout d'autres services (actions dans notre terminologie) dont les effets sont produits pour réaliser le but. Ces services peuvent eux-mêmes avoir des préconditions non soutenues causalement, ce qui induit de nouveaux raffinements, ou provoquer des réfutations avec d'autres services qui nécessitent des réparations. Nos algorithmes garantissent que ce processus converge vers un plan solution lorsqu'il existe.

6.3.2 Les règles d'interactions

La formalisation des règles d'interaction est donnée par le tableau 6.3. Pour les caractériser de manière formelle, nous définissons pour chaque acte du niveau informationnel trois sous-ensembles de règles :

1. *les règles de rationalité* qui définissent les propriétés que doit vérifier le contenu informationnel de l'acte pour être valide au sens du dialogue ;
2. *les règles du dialogue* qui spécifient les actes de dialogue autorisés pour garantir la cohérence du raisonnement produit par les agents ;
3. *les règles de mise à jour* qui précisent comment le tableau doit être modifié en fonction des actes de dialogue.

6.3.3 Les règles de contextualisation

Les règles du niveau informationnel ne suffisent pas à définir totalement les interactions entre agents. Il reste encore à préciser comment les agents débutent et clôturent la synthèse d'un plan. Nous proposons de formaliser l'ensemble des règles de contextualisation par un automate à états finis (cf. figure 6.4). Les états de l'automate définissent les états des interactions et les arêtes les différents actes de dialogues du tableau 6.2. À chaque réception ou émission d'un acte par un autre agent, l'automate tient à jour l'état courant du dialogue. Nous utilisons la notation ? et ! devant un acte de dialogue pour indiquer que l'acte est respectivement reçu (e.g., ?refine()) ou émis (e.g., !refine()) par un agent.

Essayons maintenant d'étudier plus en détail l'automate de contextualisation. Initialement, les agents sont dans un état **IDLE**. Cet état correspond à une attente par les agents d'un but à résoudre. À la réception ou à l'émission de l'acte prop.solve, le dialogue passe dans l'état **Planning** dans lequel les agents échangent des raffinements, des réfutations ou encore des réparations qui sont stockés dans leur tableau en fonction des règles définies par le niveau informationnel (cf. tableau 6.3). Cet état représente la phase de composition des différents services web. Notons que l'acte prop.solve peut être énoncé, soit par un opérateur humain, soit par un agent qui

refine(ρ, p, π) où ρ est un raffinement et p l'hypothèse raffinée du plan partiel π.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau ; (ii) p est une hypothèse formulée par π ; (iii) ρ n'a pas déjà été proposé comme raffinement de p et (iv) le plan partiel résultat π' de l'application de l'opérateur de raffinement ρ est valide au sens où ses contraintes d'ordre et d'instanciation sont cohérentes.

Dialogue : les agents peuvent raffiner toutes les hypothèses de π' ou réfuter π'.

Mise à jour : ajouter π' comme raffinement de l'hypothèse p de π dans son tableau.

refute(ϕ, π) où ϕ est une réfutation et π le plan partiel réfuté.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau et (ii) ϕ n'a pas déjà été proposée comme réfutation à l'encontre de π.

Règles : les agents peuvent réparer π.

Mise à jour : ajouter ϕ comme réfutation de π dans le tableau.

repair(ψ, ϕ, π) où ψ est une réparation de π en réponse à la réfutation ϕ.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau ; (ii) ϕ est une réfutation formulée à l'encontre de π ; (iii) ψ n'a pas déjà été proposée comme réparation de π en réponse à la réfutation ϕ et (iv) le plan partiel résultat π' de l'application de ψ est valide au sens où ses contraintes d'ordre et d'instanciation sont cohérentes.

Règles : les autres agents peuvent raffiner toutes les hypothèses de π' ou réfuter π'.

Mise à jour : ajouter π' comme réparation de la réfutation ϕ de π.

failure(Φ,π) où Φ est une menace, i.e., une hypothèse ou une réfutation de π.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau et (ii) Φ est une hypothèse ou une réfutation formulée à l'encontre de π.

Règles : $\emptyset$

Mise à jour : marquer Φ comme ne pouvant être résolue par l'agent qui a énoncé l'acte.

TABLE 6.3 – Règles d'interaction du niveau informationnel

souhaite soumettre à un groupe d'agents un but à résoudre. La phase d'ouverture du dialogue est équivalente dans les deux cas. Le contenu informationnel de l'acte prop.solve représente les buts soumis groupe d'agents. Lorsque le dialogue est ouvert, les agents sont dans l'état **Planning**. La sortie de cet état peut se produire dans deux cas :

1. *sur proposition de l'agent.* Lorsque l'agent initie une proposition de sortie sur échec (resp. sur succès), l'agent énonce l'acte prop.failure (resp. prop.success). Le dialogue passe alors dans un état **Failure** (resp. **Success**) dans lequel l'agent attend les acquittements locaux des autres agents. Si tous les agents acquittent la proposition alors les conditions de terminaison sur échec (resp. sur succès) sont vérifiées ;
2. *sur proposition d'un autre agent,* c'est-à-dire lorsqu'un agent reçoit une proposition de sortie sur échec, prop.failure, ou respectivement une proposition de sortie sur succès, prop.success. La fermeture du dialogue sur proposition d'un autre agent nécessite un découpage en deux phases :
   (a) *une phase de vérification.* Lorsqu'un de ces deux actes est reçu, le dialogue passe en instance de sortie, représenté par les états **IF** (Instance Failure) et **IS** (Instance Success). Ces deux états du dialogue correspondent à la vérification locale des propriétés de sortie de dialogue. Dans le cas d'une proposition de sortie sur échec, l'agent doit vérifier qu'il n'existe pas localement de solution au problème initialement soumis au groupe d'agents, et dans le cas d'une proposition de sortie sur succès que le plan partiel constituant le contenu informationnel de l'acte prop.success vérifie bien localement les propriétés d'un plan solution définies par la proposition 6.1 ;
   (b) *une phase d'acquittement.* Si les propriétés locales de terminaison sur échec (resp. succès) sont vérifiées alors l'agent acquitte la proposition de sortie en énonçant ack.failure (resp. ack.success), et le dialogue passe dans l'état **Failure** (resp. **Success**). Les états **Failure** et **Success** correspondent à l'attente par un agent des acquittements des autres agents. Notons que le non-acquittement par un agent de la proposition de sortie ne se traduit pas par un acte spécifique du niveau de contextualisation. En effet, supposons qu'un agent soit dans l'état **IF** (i.e., Instance Failure), si l'agent reçoit ou soumet un raffinement, une réfutation ou une réparation, cela signifie qu'un des agents est capable de poursuivre le processus de synthèse de plans. Les agents doivent par conséquent réévaluer le tableau puisque celui-ci a été modifié. De manière similaire, lorsqu'un agent se trouve dans l'état **IS** (i.e., Instance Success), si l'agent reçoit ou soumet une réfutation portant sur le plan partiel proposé comme solution, le plan partiel précédemment proposé comme solution est invalidé et l'agent doit à nouveau poursuivre le processus de synthèse de plans. Dans les deux cas, tous les agents impliqués dans le dialogue sont contraints à poursuivre le processus de synthèse de plans ce qui se traduit par un retour dans l'état **Planning** de l'automate de contextualisation.

L'automate de contextualisation garantit que la terminaison du processus de synthèse de plans ne peut avoir lieu que si l'ensemble des agents sont dans l'incapacité de faire progresser la co-construction d'un plan ou alors qu'un plan solution est présent dans le tableau. Ceci revient à calculer un état global pour déterminer si l'une des conditions de sortie est vérifiée.

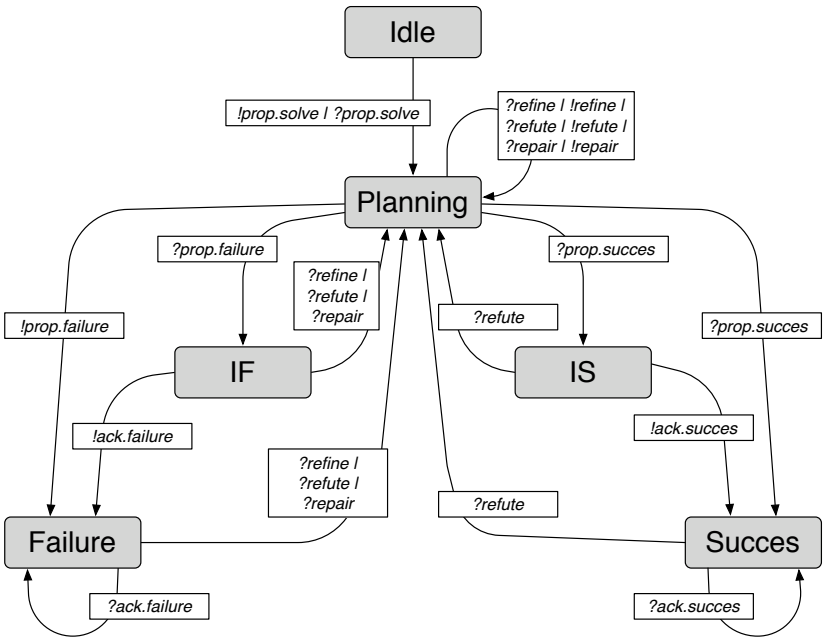


FIGURE 6.4 – Automate de contextualisation des interactions

6.3.4 La boucle de raisonnement

La définition des règles qui régissent le dialogue est nécessaire, mais ne spécifie pas les mécanismes qui guident l'interaction. Pour répondre à cette question, nous proposons d'introduire la boucle de raisonnement qui permet aux agents d'élaborer conjointement un plan. À chaque itération, un agent choisit un plan partiel déjà énoncé et cherche à lui appliquer un raffinement, une réfutation ou une réparation faisant ainsi progresser collectivement la recherche d'un plan solution. La procédure de raisonnement d'un agent peut être schématisée par l'exécution itérative des opérations suivantes :

1. Choix d'un plan partiel π non encore exploré dans son tableau ;
2. Choix de l'opération à appliquer à π, i.e., raffiner, réfuter ou réparer ;
3. Calcul des modifications à apporter à π ;
4. Soumission des modifications aux autres agents.

Le raisonnement se poursuit tant que les tableaux des différents agents ne contiennent pas au moins un plan partiel qui respecte les propriétés d'un plan solution (terminaison sur succès cf. proposition 6.1) ou alors qu'aucun agent ne peut plus modifier un plan partiel déjà proposé et ainsi faire progresser la synthèse de plans (terminaison sur échec). Lorsque l'une de ces deux conditions de terminaison est vérifiée, tous les agents mettent fin à leur raisonnement.

La boucle de raisonnement qui décrit le raisonnement d'un agent est donnée en pseudo-code par l'algorithme 10. La première opération effectuée par la boucle de raisonnement est de choisir un plan partiel à partir de son tableau. Cette sélection ne peut porter que sur les plans partiels pour lesquels l'agent peut encore proposer des raffinements, des réfutations ou des réparations. Si l'agent ne peut plus faire progresser la synthèse de plan (ligne 3), alors il doit soumettre aux autres agents une proposition de terminaison sur un échec et mettre fin à sa propre boucle de raisonnement (ligne 5). La proposition de terminaison est alors soumise aux autres agents pour acceptation.

Soulignons que le choix du plan partiel à explorer est important pour garantir de bonnes performances à l'algorithme de synthèse de plans. En effet, c'est de ce choix local que découle la politique globale d'exploration de l'espace de plans partiels. Pour plus de détails sur ces aspects, le lecteur peut se référer aux travaux de (GEREVINI et SCHUBERT, 1996).

Maintenant qu'un plan partiel a été sélectionné, l'agent doit déterminer si ce plan partiel est un plan solution. Rappelons qu'un plan partiel π est un plan solution (ligne 8) si π ne formule plus aucune hypothèse et ne peut pas être réfuté. La première propriété est vérifiée par la procédure `Open(`π`)` qui calcule l'ensemble des hypothèses présentes dans un plan partiel. Cette procédure peut être implémentée de manière efficace en utilisant une table de hachage : à chaque ajout d'une nouvelle action a au plan partiel, les hypothèses de a sont ajoutées à la table et, à chaque ajout d'un lien causal au plan partiel, les hypothèses correspondantes sont supprimées de la table. La seconde propriété est vérifiée par la procédure `Refutations(`π`)` qui calcule l'ensemble des réfutations pouvant être formulées à l'encontre du plan partiel (cf. §6.4.2). Lorsqu'un plan partiel π vérifie localement ces deux propriétés (ligne 9), l'agent doit proposer à l'ensemble des autres agents de terminer le processus de synthèse de plans en soumettant π comme plan-solution. De manière symétrique aux mécanismes de terminaison sur échec, l'agent met fin à son raisonnement. Sinon l'agent doit poursuivre son raisonnement en proposant un nouveau raffinement ou encore une nouvelle réfutation ou réparation.

Considérons tout d'abord qu'un agent décide de proposer un nouveau raffinement portant une hypothèse de π (ligne 14). L'agent calcule les raffinements possibles par l'intermédiaire de la procédure `Refine(`ϕ, π`)` qui prend en paramètres l'hypothèse ϕ à raffiner ainsi que le plan partiel π. Si aucun raffinement ne peut être proposé, alors ϕ est marquée comme ne pouvant être résolue et l'agent propage son échec. Sinon, l'agent choisit un raffinement et le soumet à la délibération des autres agents. Le même mécanisme s'applique pour les réparations (ligne 24). Si ϕ est une réfutation qui a été précédemment formulée à l'encontre d'un plan partiel, π doit être réparée. L'agent calcule alors les réparations possibles pour la réfutation grâce à la procédure `Repair(`ϕ, π`)`. Si aucune réparation n'a été produite, l'agent marque la réfutation comme ne pouvant être résolue et l'agent propage son échec. Dans le cas contraire, l'agent informe les autres agents des réparations produites.

Finalement, si ϕ est une réfutation (ligne 35), l'agent soumet simplement la réfutation aux autres agents afin de leur indiquer que dorénavant le plan partiel π n'est pas correct et doit être réparé.

Proposition 6.2. *La boucle de raisonnement est correcte et complète : chaque fois qu'il existe, du point de vue d'un agent, une solution pour $\mathcal{P} = (s_0, \mathcal{O}, g)$, cet agent termine sa boucle de raisonnement en faisant une proposition de succès avec un plan solution.*

Preuve 6.3 (Correction)**.** *Par définition, le plan partiel initial $\pi_0 = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est défini par $\mathcal{A} = \{a_0, a_\infty\}$, $\prec$ un ensemble de contraintes d'ordre cohérent et $\mathcal{I}$ un ensemble de contraintes d'instanciation cohérent. Lorsque la boucle de raisonnement fait une proposition de succès, le plan partiel π proposé ne contient plus de menace et tous les opérateurs de raffinements et de réparation maintiennent $\prec$ et $\mathcal{I}$ cohérents. Ceci est garanti par les règles du dialogue. Par conséquent, d'après la proposition 6.1, π est un plan solution.*

Le raisonnement est également correct au niveau du groupe d'agents, car la boucle de raisonnement ne s'arrête que si et seulement si un agent a initié une proposition de succès sur π (π vérifie localement les propriétés d'un plan solution pour l'agent initiateur) et que π n'est pas réfutable par les autres agents. Par conséquent, π est un plan solution pour $\mathcal{P}$ au niveau du groupe.

Algorithme 10 : Boucle de raisonnement

```
1  while raisonne = true do
2      plans ← l'ensemble des plans partiels restant à explorer
3      if plans est vide then
4          Soumettre une proposition d'échec
5          raisonne ← false
6      else
7          Sélectionner un plan partiel π ∈ plans
8          menaces ← Open(π) ∪ Refutations(π)
9          if menaces est vide then
10             Soumettre une proposition de succès portant sur π
11             raisonne ← false
12         else
13             Sélectionner une menace φ ∈ menaces
14             if φ est une hypothèse then
15                 raffinements ← Refine(φ, π)
16                 if raffinements est vide then
17                     Marquer φ comme ne pouvant être résolue
18                     Soumettre l'échec de raffinement portant sur φ
19                 else
20                     Sélectionner un raffinement ρ ∈ raffinements
21                     Soumettre ρ comme raffinement φ de π
22             else if φ est une réfutation déjà formulée à l'encontre de π then
23                 reparations ← Repair(φ, π)
24                 if reparations est vide then
25                     Marquer φ comme ne pouvant être résolue
26                     Soumettre l'échec de réparation portant sur φ
27                 else
28                     Sélectionner une réparation ψ ∈ reparations
29                     Soumettre ψ comme réparation de φ de π
30             else Soumettre φ comme nouvelle réfutation de π
```

Preuve 6.4 (Complétude)**.** *Il faut montrer qu'au moins une des traces d'exécution de la procédure de raisonnement renvoie un plan solution d'un point de vue d'un agent lorsqu'il existe. Preuve par induction, sur la longueur k d'un plan solution.*

Cas de base ($k = 0$) : *le plan vide π est solution de $\mathcal{P}$. Par conséquent, π ne comporte pas de menace et la procédure de raisonnement fait immédiatement une proposition de succès.*

Induction : *supposons que la procédure de raisonnement soit complète pour un problème nécessitant une solution de longueur k. Soit $\mathcal{P} = (s_0, \mathcal{O}, g)$ nécessitant un plan solution $\langle a_0, \ldots, a_k \rangle$ de longueur $k + 1$. Comme a_k est nécessaire pour démontrer le but g, il existe au moins un agent qui proposera de manière non déterministe un opérateur de raffinement, de réfutation ou de réparation à appliquer au plan partiel initial π_0 contenant a_k. Soit $s_{k+1} = ((s_k \cup open(a_k)) - effects^-(a_k)) \cup effects^+(a_k)$, l'état suivant l'exécution de a_k. Au prochain appel*

récursif de la procédure par les agents le plan partiel π_1 résultat de l'opérateur contenant a_k est constitué d'au moins trois actions $\{a_0, a_k, a_\infty\}$. π_1 est équivalent au plan partiel initial d'un problème défini par l'état s_0 et $g = open(a_k) \cup open(a_\infty)$. Ce problème admet une solution $\langle a_0, \ldots, a_j \rangle$ de longueur $j \leq k$. Par hypothèse d'induction, les appels récursifs de la procédure sur le plan partiel π_1 produisent une trace qui trouve le plan-solution $\langle a_0, \ldots, a_j \rangle$ et la procédure propose de sortir sur succès.

6.4 Raffiner, réfuter et réparer

Cette section présente les mécanismes nécessaires au calcul des opérateurs de raffinement, de réfutation et de réparation.

6.4.1 Les mécanismes de raffinement

L'opérateur de raffinement est utile pour démontrer qu'une hypothèse peut être vérifiée. Nous distinguons deux mécanismes de raffinement : les raffinements par ajout d'un lien causal et les raffinements par ajout d'un sous-plan partiel.

Les raffinements par ajout d'un lien causal Le premier mécanisme de raffinement consiste à ajouter un lien causal. Cette technique de raffinement est la plus simple. S'il existe déjà au sein du plan partiel π une action a_i qui a pour effet p alors il suffit d'ajouter au plan partiel π un lien causal $(a_i \xrightarrow{p} a_j)$ indiquant dorénavant que l'action a_i démontre l'hypothèse p formulée par l'action a_j. Un exemple de ce type de raffinement est donné à la figure 6.5. Le raffinement solution est un tuple constitué d'un lien causal, d'une contrainte d'ordre, $(a_i \prec a_j)$, ainsi que des contraintes d'instanciation σ nécessaires à l'unification de p avec les effets de a_i.

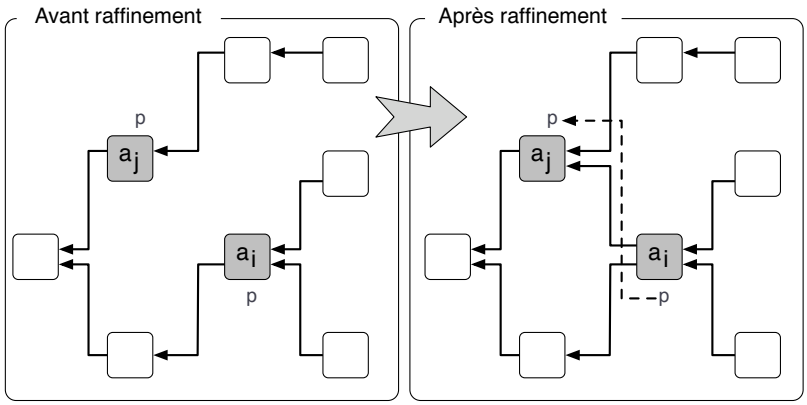


FIGURE 6.5 – Exemple de raffinement lorsqu'une action déjà contenue dans le plan partiel permet de démontrer une hypothèse p

L'intérêt de ce type de raffinement est de minimiser l'ajout de nouvelles actions au plan partiel et ainsi de réduire le nombre d'hypothèses introduites. Si l'on fait un rapprochement avec les travaux sur les interactions entre plans de (VON-MARTIAL, 1992), ce mécanisme peut être vu comme une technique de prise en compte des relations favorables[3] entre les actions proposées par les différents agents. En effet, le plan partiel construit est plus efficace puisque le raffinement conduit à la réutilisation d'une action déjà présente. On évite ainsi l'ajout d'actions redondantes au cours du processus de synthèse de plans.

3. Dans la mesure où la ressource est partageable et non consommable.

Les raffinements par ajout d'un sous-plan partiel Le second mécanisme de raffinement consiste à ajouter un sous-plan partiel pour démontrer la validité d'une hypothèse. Rappelons qu'une hypothèse p est considérée comme un sous-but à atteindre pour les autres agents. Le raffinement solution contient le sous-plan partiel produit π', un lien causal, $(a_i \xrightarrow{p} a_j)$ permettant de spécifier quelle action a_i de π' démontre l'hypothèse p, et une contrainte d'ordre $(a_i \prec a_j)$. La figure 6.6 montre un exemple de raffinement par ajout d'un sous-plan partiel pouvant lui-même contenir des hypothèses.

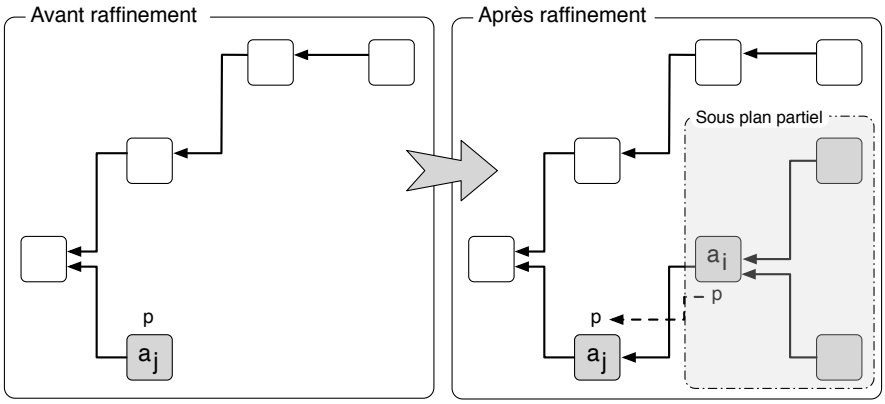


FIGURE 6.6 – Exemple de raffinement par ajout d'un sous-plan partiel

L'algorithme mis en œuvre pour produire les raffinements, i.e., les sous-plans partiels démontrant la validité d'une ou plusieurs hypothèses, repose sur une technique particulière de planification appelée *Hierarchical Task Network* (NAU et al., 2003). En comparaison des techniques de planification classiques, le principal avantage de HTN réside dans l'expressivité de son langage et ses capacités d'extension, e.g., en permettant l'ajout de procédures externes dans les préconditions des opérateurs, en ajoutant un mécanisme d'inférence etc. Cette technique permet également de modéliser et de résoudre des problèmes que les planificateurs classiques ne peuvent pas résoudre (EROL, HENDLER et NAU, 1994). En outre, avec une bonne modélisation pour guider la recherche d'un plan solution qui peut être soit partiellement soit totalement ordonné (cas considéré dans nos travaux), cette technique peut résoudre la plupart des problèmes classiques de planification beaucoup plus rapidement que des techniques telles que la recherche dans un espace d'états ou de plans. Le principal inconvénient de HTN réside dans la nécessité pour l'auteur d'un domaine de planification de spécifier non seulement les opérateurs de planification, mais également les méthodes. Ce désavantage reste toutefois assez limité dans le contexte de la spécification de services web dans la mesure où les services sont déjà décrits comme la décomposition de tâches complexes.

Dans le cadre de planification HTN, le but à atteindre n'est plus un ensemble de propriétés du monde qui doivent être vérifiées, mais une séquence d'actions primitives ou complexes à réaliser à partir de l'état initial de croyance des agents. La production d'un plan peut alors s'exprimer comme la décomposition récursive d'actions complexes en sous-actions jusqu'à ce que toutes les actions soient des actions *primitives*, i.e., pouvant être exécutées en appliquant un opérateur. Pour passer de la représentation d'un but défini comme un ensemble de propositions à celle d'une action à réaliser, il suffit de décrire une méthode particulière qui prend en paramètre le but à atteindre.

La définition d'un raffinement peut s'exprimer de la manière suivante.

Définition 6.10 (Raffinement). *Soit un problème de planification* $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$. *Un plan partiel* π *est un raffinement de* $\mathcal{P}$ *si toutes les linéarisations* $\lambda \in$ *completion*(π) *raffinent* $\mathcal{P}$. *Soit* $\lambda = \langle a_0, \ldots, a_k \rangle$ *une séquence d'actions primitives,* λ *raffine* $\mathcal{P}$ *si l'une des conditions suivantes est vérifiée :*

Cas 1 : *la séquence d'actions à réaliser est vide alors* λ *est la séquence vide ;*

Cas 2 : α_0 *est primitive :* α_0 *est identique à* a_0, α_0 *est applicable à partir de* s_0 *et* $\lambda = \langle a_1, \ldots, a_k \rangle$ *raffine* $(s_1, \mathcal{O}, \mathcal{M}, \langle \alpha_1, \ldots, \alpha_n \rangle)$ *;*

Cas 3 : α_0 *est complexe : il existe une expansion* $\langle r_1, \ldots, r_j \rangle$ *applicable pour réaliser* α_0 *à partir de* s_0 *et* λ *raffine* $(s'_0, \mathcal{O}, \mathcal{M}, \langle r_1, \ldots, r_j, \alpha_1, \ldots, \alpha_n \rangle)$ *avec* $s'_0 = s_0 \cup$ *open*(α_0).

La procédure qui produit les raffinements se découpe en deux phases : l'expansion de l'arbre de raffinements (détaillée dans la suite de cette section) et l'extraction d'un plan partiel qui consiste à parcourir une branche de l'arbre d'une feuille solution au nœud racine en initialisant les actions, les contraintes d'ordre, les contraintes d'instanciation et les liens causaux du raffinement. La construction de l'arbre de raffinements consiste à décomposer récursivement les actions complexes contenues dans les nœuds de l'arbre jusqu'à ce qu'une feuille soit produite, i.e., un nœud possédant une séquence d'actions vide. Cette feuille représente l'état du monde qui sera atteint après l'exécution du raffinement.

L'algorithme d'expansion de l'arbre de raffinements s'appuie sur la définition 6.10. Initialement, la procédure d'expansion de l'arbre est exécutée avec le nœud représentant l'état initial des croyances de l'agent s_0 et la séquence des actions à réaliser $\langle \alpha_0, \ldots, \alpha_n \rangle$. La procédure commence par tester si $\langle \alpha_0, \ldots, \alpha_n \rangle$ est la séquence vide. Dans ce cas, une feuille de l'arbre est atteinte et le nœud n est retourné comme nœud solution (cf. cas 1 définition 6.10). Sinon la procédure essaie de réaliser la première action α_0. Deux cas sont à envisager selon que α_0 est :

1. *une action primitive.* Pour chaque opérateur contenu dans $\mathcal{O}$, la procédure teste s'il peut réaliser l'action α_0. Si c'est le cas, la procédure calcule les substitutions unifiant les préconditions de l'opérateur avec l'état courant s contenu dans le nœud n. Finalement, pour chaque substitution σ, l'algorithme ajoute un nœud fils au nœud n (cf. cas 2 définition 6.10) ;
2. *une action complexe.* La procédure repose sur le même principe que pour une action primitive. Cependant elle teste cette fois, non plus les opérateurs, mais les méthodes qui permettent de réaliser α_0 et qui sont applicables dans s. Pour chacune de ces méthodes un nouveau nœud fils est ajouté au nœud n (cf. cas 3 définition 6.10).

Finalement, la procédure choisit de manière non déterministe un nouveau nœud parmi les nœuds fils de n et s'appelle récursivement avec comme paramètre le nouveau nœud sélectionné.

6.4.2 Les mécanismes de réfutation

En toute généralité, les mécanismes de réfutation cherchent à démontrer qu'un plan partiel est incorrect. Dans notre approche, un plan partiel est incorrect s'il contient des séquences d'actions non valides. Nous avons défini une réfutation (cf. définition 6.9) comme un couple $(a_k, a_i \xrightarrow{p} a_j)$ indiquant que l'action a_k a pour effet q tel que q supprime une précondition p nécessaire à l'exécution de a_j. Autrement

dit, le calcul des réfutations consiste à déterminer les actions a_k qui invalident un lien causal $(a_i \xrightarrow{p} a_j)$ (cf. figure 6.3).

6.4.3 Les mécanismes de réparation

Les mécanismes de réparation calculent les modifications à apporter aux plans partiels lorsque ceux-ci ont été préalablement réfutés. Nous distinguons les réparations par ajout de contraintes d'ordre et par ajout d'un sous-plan partiel.

Les réparations par ajout de contraintes d'ordre. Le premier type de réparation consiste à introduire une contrainte d'ordre entre a_k et les actions a_i et a_j. En effet, si une réfutation a été formulée à l'encontre d'un plan partiel, c'est parce que a_k supprime une propriété du monde nécessaire à l'exécution de a_j. Il faut donc s'assurer que a_k ne puisse pas être exécutée entre les actions a_i et a_j, i.e., $(a_i \prec a_k \prec a_j)$. Pour garantir que cela ne peut pas se produire, il faut ajouter soit une contrainte d'ordre $(a_j \prec a_k)$, imposant que l'exécution de a_k soit réalisée après a_j, soit une contrainte d'ordre $(a_k \prec a_i)$, imposant que a_k s'exécute avant a_i (cf. figure 6.7). L'existence préalable de contraintes d'ordre dans le plan partiel entre l'action a_k et les actions a_j et a_i contraignent les choix possibles. Le tableau 6.4 récapitule les réparations par ajout de contraintes d'ordre en fonction des contraintes qui lient a_k avec les autres actions.

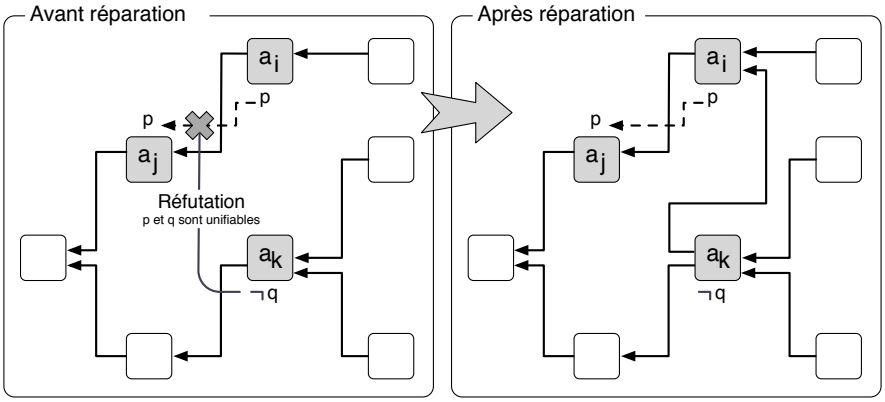


FIGURE 6.7 – Exemple de réparation par ajout de contraintes d'ordre

Contraintes existantes sur a_k	Contraintes à ajouter
$\emptyset$	$a_j \prec a_k$ ou $a_k \prec a_i$
$a_k \prec a_j$	$a_k \prec a_i$
$a_i \prec a_k$	$a_j \prec a_k$
$a_i \prec a_k$ et $a_k \prec a_j$	pas de solution

TABLE 6.4 – Taxonomie des réparations par ajout d'une contrainte d'ordre

Les réparations par ajout d'un sous-plan partiel. Étant donné que l'action a_k supprime une propriété p nécessaire à l'exécution de l'action a_j, la réparation consiste à ajouter un sous-plan partiel permettant de produire p après l'exécution de a_k. En ce

sens, les réparations par ajout de sous-plans partiels peuvent être vues comme le raffinement d'une hypothèse implicite formulée par l'action a_k. En effet, en proposant l'action a_k, un agent suppose qu'un autre agent sera capable de recréer la propriété q. Par conséquent, ce type de réparation qui repose sur la production de raffinements vérifiant p (cf. §6.4.1) participe à la mise en œuvre de la coopération entre les agents.

Exemple 6.4. Considérons l'exemple donné par la figure 6.8. L'action a_k réfute le lien causal $(a_i \xrightarrow{p} a_j)$ car l'effet p est unifiable avec q. L'ajout du sous-plan partiel (ne contenant que l'action a_l) produit l'effet q après sa suppression par a_k. Par conséquent, q est vérifiée dans l'état précédant a_j (a_j peut être exécutée) et a_k ne réfute plus $(a_i \xrightarrow{p} a_j)$.

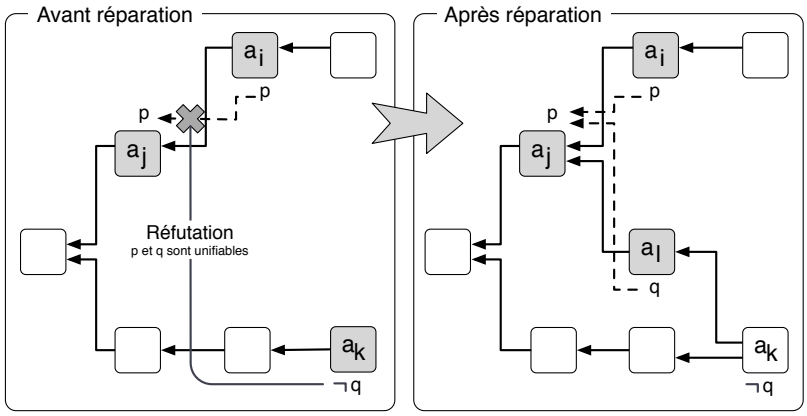


FIGURE 6.8 – Exemple de réparation par ajout d'un sous-plan partiel

6.5 Conclusion

Dans ce chapitre, nous avons présenté un modèle de planification entièrement distribué dans lequel les agents raisonnent conjointement sur leurs compétences et leurs croyances respectives pour atteindre un but commun prédéfini. Au niveau multi-agent, la synthèse de plans est vue comme un raisonnement collectif et révisable fondé sur une recherche dans un espace de plans partiels où les agents échangent des raffinements, i.e., des plans qui peuvent être exécutés si certaines conditions sont vérifiées, et des réfutations, i.e., des objections quant à la réalisation du plan. Au niveau agent, le modèle repose sur leur capacité à produire des raffinements pouvant être contradictoires avec leurs croyances en s'appuyant sur une technique de planification hiérarchique modifiée à cet effet.

L'avantage de l'approche réside dans le fait qu'elle intègre les différentes phases de la planification distribuée (cf. figure 4.6) en permettant aux agents de fusionner leurs compétences et leurs croyances hétérogènes. Par ailleurs, le modèle propose de tirer parti de l'efficacité et de l'expressivité des techniques de planification hiérarchiques pour spécifier les problèmes de planification et avec l'expressivité des plans produits par les techniques de planification partiellement ordonnée. De notre point de vue, cette approche est adaptée aux applications pour lesquelles les agents sont coopératifs et pour lesquelles les problèmes de planification à résoudre sont fortement inter-dépendants.

Le travail présenté ouvre pour nous en l'état deux pistes de travail. Tout d'abord, nous souhaitons développer des heuristiques pour guider de manière plus efficace à la fois la recherche d'un plan solution au niveau multi-agent mais aussi pour guider

au niveau local le choix des préconditions à relaxer. Les travaux récents (STOLBA, FISER et KOMENDA, 2015, 2016) nous encouragent dans ce sens. Finalement, nous souhaitons étendre notre modèle à la planification en boucle fermée, comme présenté dans la première partie de manuscrit, en généralisant le concept de buts ouverts à des propriétés découvertes pendant l'exécution du plan et le concept de réfutation à des observations provenant de l'environnement.

Chapitre 7

Planification multi-agent appliquée au problème de poursuite-évasion

Ces dernières années s'est développé un intérêt croissant pour l'usage d'approches multi-agents appliquées à la robotique dans de nombreux contextes applicatifs, e.g., l'agriculture (Tokekar, Mulla et Isler, 2016), la santé (Gautham et al., 2015), l'industrie (Digani et al., 2014), etc. Contrairement aux chapitres précédents, nous proposons ici d'aborder le problème de la planification distribuée sous un angle domaine spécifique en nous intéressant au problème de la planification distribuée de trajectoires pour la résolution du problème de poursuite-évasion (Benda, Jagannathan et Dodhiawala, 1986) aussi connu sous le nom de proie-prédateurs. Le problème consiste à planifier les trajectoires d'un groupe de robots dans un environnement afin de garantir qu'aucun intrus n'y est caché. Dans sa version originelle, les agents (i.e, la proie et les prédateurs) évoluent dans un environnement modélisé par une grille. À chaque unité de temps, les agents peuvent se déplacer horizontalement ou verticalement d'une case. Le but des prédateurs est de *capturer la proie* en l'encerclant en un minimum de temps. Les limites des approches développées pour ce problème apparaissent dès lors que l'on envisage un environnement réaliste et composé d'obstacles.

L'une des principales difficultés de ce problème, comparé avec la *simple* exploration d'un environnement, réside dans le fait qu'un agent intrus peut se déplacer dans une zone de l'environnement précédemment explorée par les poursuivants (cf. figure 7.1). Les algorithmes permettant de résoudre ce problème peuvent trouver de larges champs d'application dans de nombreux domaines comme la robotique pour la localisation d'objectifs divers (e.g., recherche de mines ou surveillance d'un chenal par des engins sous-marins autonomes, exploration d'environnements hostiles par des drones ou des robots terrestres dans le cadre d'une bulle opérationnelle aéroterrestre, etc.). Au-delà de la robotique, ces algorithmes peuvent aussi être mis en œuvre pour contrôler le déplacement des personnages d'un jeu vidéo ou d'avatars dans un monde virtuel.

Le chapitre est organisé de la façon suivante. Dans une première partie, nous proposons un bref état de l'art sur le problème de poursuite-évasion. Dans un second temps, nous proposons une formalisation du problème de poursuite-évasion abordé dans ce chapitre. La troisième partie présente notre contribution (Pellier et Fiorino, 2005a) pour résoudre le problème de poursuite-évasion dans des environnements 2D inconnus par des robots poursuivants collaboratifs. Finalement, nous terminons par une évaluation expérimentale de notre contribution.

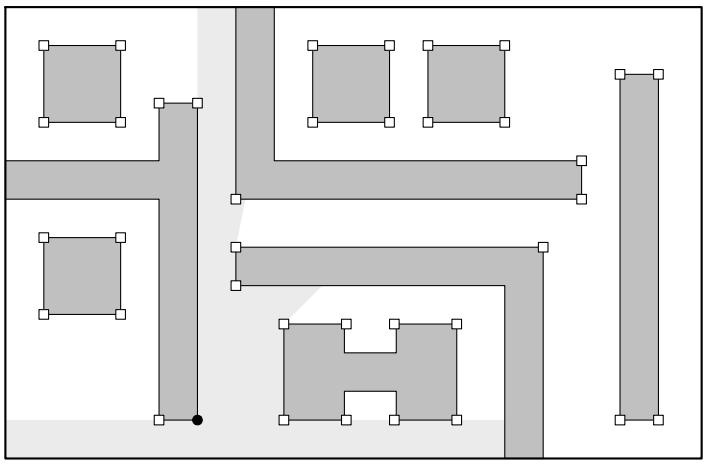


FIGURE 7.1 – Exemple d'environnement labyrinthique : Le point noir représente un agent poursuivant, la zone gris clair sa vision courante de l'environnement, les zones gris foncé les obstacles de l'environnement et les carrés blancs les points critiques caractéristiques de l'environnement

.

7.1 État de l'art sur le problème de poursuite-évasion

Le problème de *poursuite-évasion* a été abordé sous de nombres angles, notamment celui de la théorie des jeux, de la théorie des graphes, de la géométrie computationnelle et de la robotique. Concernant la théorie des jeux, (LEVY et ROSENSCHEIN, 1992) a considéré le problème de poursuite-évasion dans le cadre d'agents coopératifs évoluant dans un environnement rudimentaire (i.e., une grille). Un certain nombre de résultats ont également été démontrés dans le cadre de la théorie des graphes (l'environnement est représenté par un graphe), dans lesquels les poursuivants et les intrus se déplacent de sommet en sommet jusqu'à ce qu'un poursuivant se trouve sur le même sommet que l'intrus. On peut citer en particulier les travaux précurseurs de (PARSONS, 1976) et les travaux plus récents de (KEHAGIAS, HOLLINGER et SINGH, 2009). Dans le domaine de la robotique, cadre du travail présenté dans ce chapitre, le problème de poursuite-évasion a été introduit par (SUZUKI et YAMASHITA, 1992). Depuis, de nombreuses variations du problème ont été étudiées pour prendre en compte les caractéristiques de l'environnement, des poursuivants ou encore celles de l'intrus à débusquer. Considérons tout d'abord l'environnement dans lequel évoluent les robots. Certains travaux font l'hypothèse qu'il est polygonal 2D (BHADAURIA et ISLER, 2011 ; GUIBAS et al., 1999) ou encore 3D (LAZEBNIK, 2001). En ce qui concerne les caractéristiques des poursuivants, une première famille de travaux cherche à prendre en compte le caractère imparfait de leurs capteurs en développant des algorithmes capables de fonctionner avec une capacité de perception limitée (MURRIETA-CID et al., 2007) ou bruitée (HOLLINGER, KEHAGIAS et SINGH, 2007). Une seconde famille s'est focalisée sur leur capacité de déplacement, par exemple sur le fait qu'ils soient statiques (CRASS, SUZUKI et YAMASHITA, 1995) ou mobiles dans l'environnement avec une vitesse limitée (TOVAR et LAVALLE, 2006). Finalement, une dernière famille de travaux s'est intéressée au développement d'approches multi-robots avec des robots aux compétences homogènes ou au contraire hétérogènes (HOLLINGER, SINGH et KEHAGIAS, 2010 ; HOLLINGER et al., 2009). Pour terminer, de nombreux travaux ont

également porté sur la prise en compte des caractéristiques de l'intrus. Par exemple, certains travaux font l'hypothèse qu'il est capable de se déplacer à une vitesse plus ou moins importante (RAMANA et KOTHARI, 2017) ou encore qu'il n'y a pas qu'un seul intrus (LAU, S. et DISSANAYAKE, 2005) à débusquer. Cette catégorisation est très schématique. De nombreux travaux existent à l'interface et croisent la prise en compte des caractéristiques précédemment citées. Nous pouvons renvoyer le lecteur à l'article d'état de l'art proposé par (CHUNG, HOLLINGER et ISLER, 2011) pour une présentation plus détaillée.

7.2 Formalisation du problème

Notre contribution s'inscrit dans le cadre du problème de poursuite-évasion en environnement 2D dans un contexte multi-robots (cf. figure 7.1). Nous ne considérons que le cas d'obstacles polygonaux ce qui n'entraîne aucune perte de généralité par rapport au cas plus réaliste d'un environnement courbe (LAVALLE et HINRICHSEN, 2001). Les poursuivants ont des compétences homogènes. Leur perception est illimitée omnidirectionnelle. Leur déplacement est continu et leur vitesse limitée. Le nombre d'intrus ainsi est indéterminé et leur vitesse de déplacement est illimitée et continue. L'originalité de notre approche est de considérer que l'environnement est *a priori* inconnu des poursuivants. La principale contribution réside dans la mise en œuvre d'un algorithme de coopération entre les poursuivants afin d'organiser une véritable exploration coordonnée de l'environnement inconnu afin d'optimiser le nombre de poursuivants nécessaires ainsi que leurs trajectoires afin de garantir l'absence d'intrus.

Les agents poursuivants et intrus sont modélisés par des points dans un espace euclidien 2D. Soit F l'ensemble des points de l'espace qui n'appartiennent à aucun obstacle. Tous les poursuivants et les intrus doivent se trouver dans F. Soit $e(t) \in F$, la position de l'intrus au temps $t \geq 0$. Nous supposons que $e : [0, \infty) \rightarrow F$ est une fonction continue, et que l'intrus est capable de se déplacer à une vitesse arbitraire élevée (éventuellement infinie). La position initiale de l'intrus $e(0)$ et sa trajectoire e sont supposées inconnues des poursuivants.

Soit $\tau^i(t)$, la position du i^e poursuivant à l'instant $t \geq 0$. $\tau^i : [0, \infty) \rightarrow F$ représente la trajectoire continue du i^e poursuivant. Soit τ, l'ensemble des trajectoires continues des N poursuivants : $\tau = \{\tau^1, \ldots, \tau^N\}$. Pour tout point $q \in F$, soit $V(q)$ l'ensemble de tous les points visibles de F en q (i.e., tous les segments reliant q à un point de $V(q)$ sont strictement inclus dans F). La ronde τ est une solution si pour chaque fonction continue $e : [0, \infty) \rightarrow F$, il existe un instant $t \in [0, \infty)$ et $i \in \{1, \ldots, N\}$ tel que $e(t) \in V(\tau^i(t))$. Ceci implique que l'intrus ne peut pas échapper à ses poursuivants : il sera nécessairement débusqué à un instant donné.

La formalisation du problème de poursuite-évasion met en lumière deux difficultés. La première est de trouver une trajectoire pour chaque poursuivant $\tau^i(t)$ avec $i \in \{1, \ldots, N\}$ telle que τ^i garantit qu'aucun intrus n'est dans l'environnement en considérant toutes les trajectoires possibles des intrus et sachant qu'ils peuvent se déplacer vers des zones de l'environnement déjà explorées. La seconde difficulté réside dans le calcul du nombre minimum de poursuivants $H(F)$ requis pour garantir que les intrus seront débusqués. Le calcul de $H(F)$ est NP-diffile. Les travaux de (BORIE, TOVEY et KOENIG, 2011) donne une borne dans le pire des cas sur le nombre de poursuivants nécessaires : $H(F) = O(\sqrt{h} + log\ n)$ ou n est le nombre d'arêtes de F et h le nombre de zones de F qui sont simplement connectées.

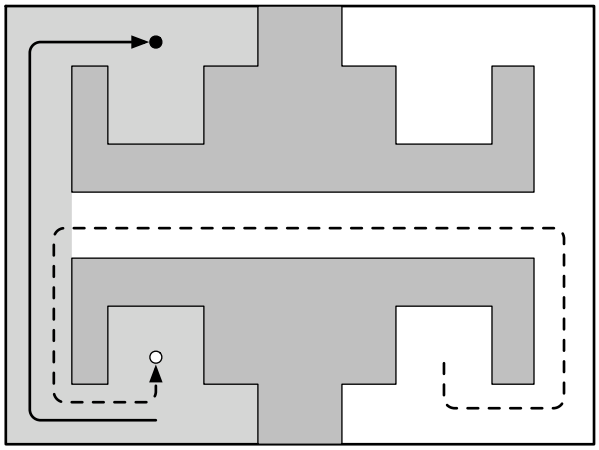


FIGURE 7.2 – Un intrus (le point blanc) peut se déplacer dans une zone déjà explorée (la zone grisée) par un poursuivant (le point noir)

Dans la suite de ce chapitre, nous supposerons que l'environnement n'est pas connu des poursuivants. Par conséquent, une phase d'exploration est nécessaire préalablement à toute recherche d'intrus dans l'environnement. De plus, nous cherchons à résoudre le problème en minimisant le nombre de poursuivants nécessaires. Lorsque le nombre de poursuivants n'est pas suffisant pour surveiller l'environnement, nous supposons qu'il est possible d'ajouter incrémentalement un autre poursuivant dans l'environnement. Tout l'enjeu est de proposer des mécanismes de coopération pour minimiser le nombre de poursuivants nécessaires.

7.3 Principe général de notre approche

Détecter un intrus dans un environnement clos est une tâche complexe même dans des environnements relativement simples. En effet, l'intrus peut se déplacer et se cacher dans une zone de l'environnement déjà explorée (cf. figure 7.2). Par conséquent, la trajectoire des poursuivants doit être calculée en tenant compte de toutes les trajectoires possibles des intrus pour les empêcher de se déplacer dans des zones déjà explorées sans être vus. Nous appellerons les zones de F qui sont susceptibles de contenir un intrus des zones *contaminées* et par opposition les zones dans lesquelles les poursuivantes sont certains qu'aucun intrus n'est présent des zones *nettoyées*. L'objectif des agents poursuivants consiste donc à décontaminer toutes les zones de F qui peuvent potentiellement cacher un intrus. Pour raisonner, les poursuivants ont besoin d'une représentation topologique de l'environnement pour planifier leur trajectoire, mais aussi pour raisonner sur celles des éventuels intrus et d'une représentation de l'état d'avancement de la décontamination de leur environnement. Pour répondre à cette double nécessité, notre approche repose sur deux graphes : un graphe de navigation que nous notons G_n et un graphe de poursuite que nous notons G_p.

7.3.1 Le graphe de navigation

Seuls quelques points caractéristiques de l'espace libre F, appelés *points critiques*, sont pertinents pour résoudre le problème de poursuite-évasion.

Définition 7.1 (Point critique)**.** *Un sommet de F est un point critique critique si et seulement si l'angle formé par ses côtés adjacents est égal ou supérieur à π.*

Ces points caractéristiques sont pertinents dans la mesure où si l'on place un poursuivant avec une vision omnidirectionnelle à chacun des points critiques de F alors on garantit qu'aucun intrus ne peut s'y cacher. Dans la suite, nous faisons l'hypothèse que tout poursuivant est capable de détecter ces points critiques dans l'environnement. Cette hypothèse est réaliste. Des solutions techniques peuvent assez simplement être implémentées en utilisant un LIDAR (*Laser Detection And Ranging*). De plus, nous supposerons qu'un poursuivant est capable de se déplacer entre les différents points critiques. L'ensemble des mouvements possibles est modélisé sous la forme d'un *graphe de navigation*. Un exemple de graphe de navigation est donné à la figure 7.4(a). Il correspond à l'environnement de la figure 7.3.

Définition 7.2 (Graphe de navigation)**.** *Un graphe de navigation est un graphe orienté $G_n = (V, E)$ où V est l'ensemble des points critiques de F et E l'ensemble des arcs de G. Il existe un arc entre deux points critiques si les deux points critiques sont mutuellement visibles.*

7.3.2 Le graphe de poursuite

Les poursuivants doivent posséder également une représentation de l'état d'avancement de la décontamination de l'environnement, i.e., pour chaque zone de l'environnement modélisée par G_n, lesquelles sont contaminées ou nettoyées. Cette représentation s'appuie sur ce que nous appelons le graphe de poursuite. Chaque nœud du graphe représente un état de la poursuite à un instant donné de la ronde d'un poursuivant. Un état est constitué de la liste des points critiques de l'environnement auxquels on associe une étiquette : contaminés ou nettoyés. Les arcs du graphe de poursuite décrivent les déplacements qui permettent d'atteindre un état de poursuite. Pour un poursuivant, la recherche d'une solution consiste donc à trouver un chemin dans le graphe de poursuite à partir d'un état initial de poursuite [1] et de sa position initiale dans l'environnement pour atteindre un état dans lequel tous points critiques sont nettoyés. Chaque agent possède son propre graphe de poursuite. Il est par conséquent nécessaire que les poursuivants échangent des informations pour élaborer une vue globale de la poursuite. Un exemple de graphe de navigation est donné à la figure 7.4(b).

Définition 7.3 (Graphe de poursuite)**.** *Un graphe de poursuite est un graphe orienté $G_p = (V, E)$ où V est l'ensemble des états de poursuite atteignables à partir du graphe de navigation G_n modélisant l'environnement et E l'ensemble des arcs de G_p. Il existe un arc entre deux sommets v_1 et v_2 de G_p s'il existe un arc dans le graphe de navigation G_n qui permette de construire v_2 à partir de v_1. Chaque arc est étiqueté par la distance euclidienne entre v_1 et v_2.*

7.3.3 L'algorithme mono-agent en environnement connu

L'algorithme calcule la trajectoire optimale en termes de distance parcourue. Nous supposons que la position initiale du poursuivant est donnée en entrée de l'algorithme. Il comporte 4 étapes :

Étape 1 (Calcul des points critiques)

La première étape consiste à déterminer les points critiques de l'environnement (cf. figure 7.3(e)) : les points critiques sont étiquetés de S_1 à S_5). L'identification des

1. Initialement, tous les points critiques sont considérés comme contaminés.

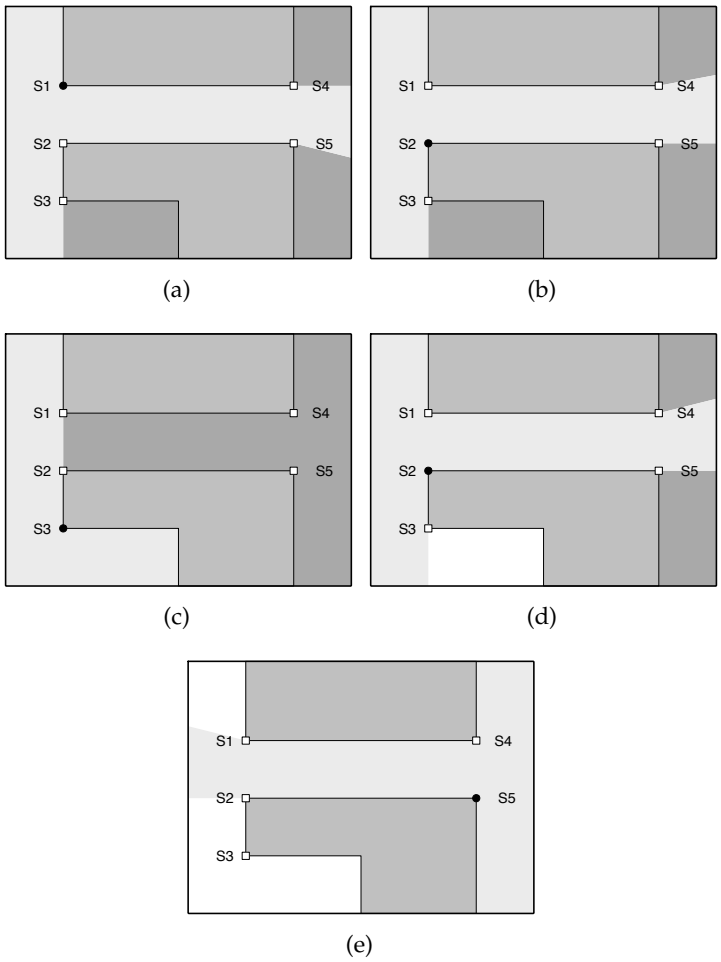


FIGURE 7.3 – Le poursuivant est représenté par un point noir. La zone gris clair représente la perception du poursuivant. Les carrés blancs représentent les points critiques étiquetés de S_1 à S_5. Les zones blanches représentent les zones nettoyées de l'environnement tandis que les zones gris foncé sont les zones qui restent à décontaminer.

points critiques est linéaire en fonction du nombre de segments des obstacles de l'environnement.

Étape 2 (Construction du graphe de navigation)

L'algorithme construit ensuite le graphe de navigation de l'environnement sur la base des points critiques détectés lors de l'étape 1 (cf. figure 7.4(a)). L'algorithme relie tous les sommets mutuellement visibles. Le graphe de navigation est connecté si l'environnement l'est. Chaque arc est étiqueté par la distance euclidienne entre les deux sommets reliés. Le graphe de navigation construit représente tous les mouvements possibles d'un poursuivant.

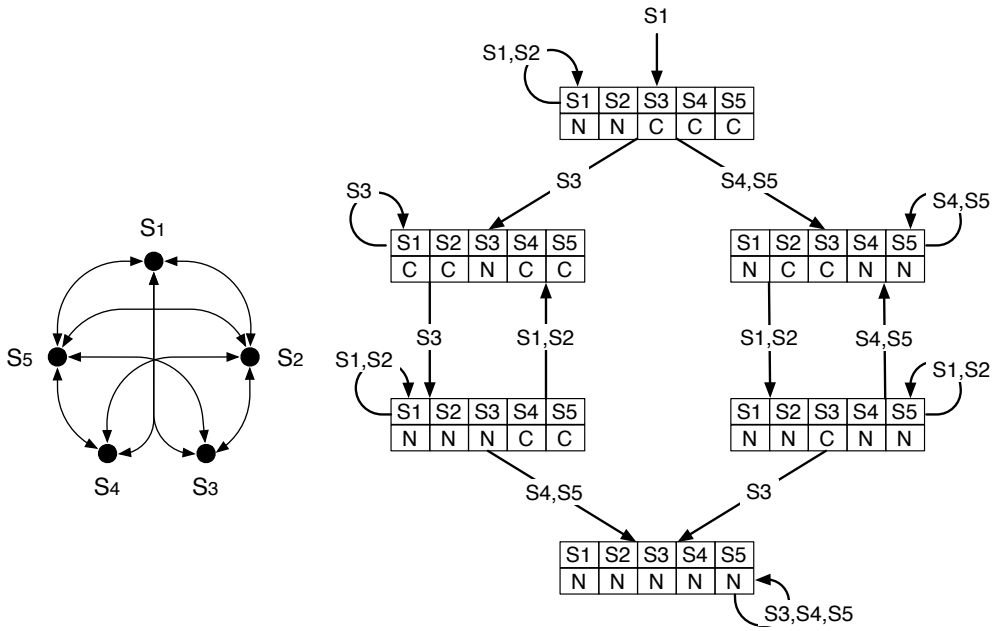


FIGURE 7.4 – a) le graphe de navigation et b) le graphe de poursuite correspondant à l'environnement de la figure 7.3 : "N" signifie nettoyé et "C" contaminé.

Étape 3 (Construction du graphe de poursuite)

La troisième étape consiste à construire le graphe de poursuite (cf. figure 7.4(b) à partir du graphe de navigation. L'état initial de poursuite est calculé à partir de la position initiale du poursuivant dans l'environnement. Soit p_1 la position du poursuivant, un point critique p_2 d'un nœud du graphe de poursuite est considéré comme nettoyé si p_2 est visible depuis p_1 et les deux segments adjacents à p_2 sont visibles entièrement depuis p_1. Dans le cas contraire, le point critique est considéré comme contaminé. La procédure se répète pour tous les points critiques de l'environnement. Maintenant que l'état initial de poursuite a été construit, l'algorithme peut construire pour chaque déplacement possible à partir de la position initiale du poursuivant et du graphe de navigation l'ensemble des états de poursuite atteignables. La création des états de poursuite repose sur la même procédure que pour la création de l'état initial de poursuite. Toutefois, deux conditions supplémentaires doivent être respectées. En effet, il faut vérifier qu'un point critique précédemment nettoyé n'est pas contaminé une nouvelle fois au cours du déplacement de p_1 à p_2. Il faut donc vérifier que :

1. pour chaque point critique nettoyé dans l'état courant de poursuite en position p_1, qu'il n'existe pas de chemin partant d'un point critique contaminé au point critique considéré ;
2. Si un tel chemin existe, il faut vérifier qu'au moins un segment du chemin reste toujours visible pendant le déplacement de p_1 et p_2.

Si ces deux conditions sont vérifiées alors le point critique reste nettoyé. La procédure de construction du graphe de poursuite garantit que si un état de poursuite solution, i.e., dont tous les points critiques sont nettoyés existe, alors il sera construit.

Étape 4 (Calcul de la trajectoire optimale)

La dernière étape consiste à calculer le chemin le plus court entre l'état initial de poursuite et l'état solution. Nous utilisons ici l'algorithme Dijsktra. La figure 7.3 montre la trajectoire (S_1, S_2, S_3, S_2, S_5) calculée par l'algorithme pour un poursuivant.

7.3.4 L'algorithme mono-agent en environnement inconnu

L'algorithme présenté suppose que l'environnement est complètement connu et que le graphe de navigation peut être construit préalablement à la recherche de la trajectoire optimale dans le graphe de poursuite. Dans le cas où l'environnement n'est pas connu, le poursuivant doit alterner une phase d'exploration de l'environnement pour construire une représentation de l'environnement, i.e., compléter et mettre à jour son graphe de navigation, et une phase de surveillance en s'appuyant sur la partie de l'environnement exploré et dorénavant connu.

L'algorithme d'exploration

L'objectif de l'algorithme d'exploration est de construire le graphe de navigation (cf. Algo. 11). L'algorithme d'exploration fonctionne sur le principe d'une exploration en largeur d'abord afin de garantir une exploration exhaustive de proche en proche. L'algorithme prend en entrée : (1) la position initiale p du poursuivant dans l'environnement, (2) le graphe de navigation G_n représentant l'environnement connu par le poursuivant (initialement le graphe de navigation est vide) et (3) une borne sur la profondeur d'exploration permettant de limiter l'exploration. L'algorithme met à jour la position courante du poursuivant ainsi que le graphe de navigation et retourne la trajectoire d'exploration exécutée.

Tant que la limite d'exploration n'est pas atteinte (ligne 3), le poursuivant récupère les points critiques identifiés comme non explorés à partir de son graphe de navigation (ligne 4) avant de les explorer un par un (ligne 5). La fonction de déplacement implémentée calcule la trajectoire la plus courte de la position courante du poursuivant au point critique en cours d'exploration (ligne 6). À chaque exploration d'un nouveau point critique (ligne 7), le graphe de navigation est mis à jour à partir des nouveaux points critiques perçus par le poursuivant à sa nouvelle position (ligne 8). À chaque nouvelle exploration, la trajectoire (ligne 9) et la position courante du poursuivant (ligne 10) sont mises à jour.

L'algorithme de poursuite

L'algorithme complet de poursuite pour un poursuivant (cf. Algo. 12) alterne exploration, découpage de l'environnement en zones qui peuvent être surveillées par d'autres poursuivants lorsque celui-ci ne peut être surveillé seul et poursuite de l'intrus. L'algorithme prend en entrée : (1) le graphe de navigation G_n représentant l'environnement connu du poursuivant ; (2) sa position courante p ; (3) l'état actuel de la surveillance s, indiquant pour chaque point critique exploré s'il est contaminé ou nettoyé, et (4) une borne sur la profondeur d'exploration. L'algorithme retourne la trajectoire exécutée par le poursuivant. Le graphe de navigation, la position du poursuivant ainsi que l'état de poursuite sont des paramètres de sortie mis à jour par l'algorithme. Lors du premier appel à la procédure de poursuite, le graphe de navigation est initialisé à partir de la position initiale du poursuivant et des points critiques visibles depuis cette position (ligne 2).

Algorithme 11 : Explore(p, G_n, *cutoff*)

```
1  Let τ an empty trajectory
2  depth ← 0
3  while depth < cutoff do
4      points ← UnexploredCriticalPoints(G_n)
5      forall p' ∈ points do
6          τ' ← PlanTrajectory(p, p', G_n)
7          Execute(τ')
8          Update(G_n, VisibleCriticalPointsFrom(p))
9          τ ← τ ⊕ τ'
10         p ← p'
11     depth ← depth + 1
12 return τ
```

L'algorithme s'articule autour d'une boucle principale (ligne 3) qui prend fin lorsque tous les points critiques de l'environnement ont été explorés. Dans cette boucle, le poursuivant commence par explorer l'environnement en utilisant la procédure d'exploration décrite précédemment (ligne 4) puis il met à jour l'état courant de la poursuite. Cette mise à jour s'opère en respectant les conditions de contamination et de nettoyage des points critiques définies à l'étape 3 de l'algorithme de poursuite en environnement connu (cf. §7.3.3). Si l'état de poursuite obtenu est un état terminal solution, i.e., tous les points critiques de l'environnement sont nettoyés, alors l'algorithme se termine et la trajectoire exécutée par le poursuivant est retournée (ligne 7). Dans le cas contraire, l'algorithme essaie de nettoyer la zone de l'environnement qu'il vient d'explorer. Il construit alors un graphe de poursuite appliquant la procédure pour un poursuivant en environnement connu. Deux cas sont alors à considérer :

1. Si le graphe de poursuite contient un état ou tous les points critiques sont nettoyés (ligne 10), il extrait du graphe de poursuite la trajectoire d'optimale de poursuite (ligne 11), l'exécute (ligne 13) et met à jour son état de poursuite (ligne 14).
2. Sinon, cela signifie que le poursuivant seul ne peut résoudre le problème qui lui est assigné. Il est donc dans l'obligation de demander l'assistance d'autres poursuivants (ligne 15).

La coopération entre les poursuivants repose sur leur capacité à découper récursivement l'environnement en zones pouvant être surveillées par un unique poursuivant en positionnant un poursuivant à un point critique stratégique de l'environnement. Cette décomposition est réalisée de la manière suivante. Tout d'abord, le poursuivant choisit un point critique comme point de coupe (ligne 16). Ce point est choisi comme le nœud du graphe de navigation possédant le degré le plus grand. En cas d'égalité, un tirage aléatoire permet de n'en choisir qu'un. En choisissant le nœud avec le degré le plus élevé, l'idée est de maximiser les chances que le découpage de l'environnement en ce point conduise à un découpage de l'environnement en zones indépendantes, i.e., conduise après suppression du point critique au découpage du graphe de navigation en plusieurs composantes connexes (ligne 17). Il s'agit ensuite de déterminer le poursuivant qui pourra se positionner au point critique choisi. Ce choix est réalisé après délibération avec les autres poursuivants

(ligne 18). Nous détaillons cette étape au paragraphe suivant. Finalement, l'algorithme de poursuite est relancé récursivement sur pour chaque zone de l'environnement (lignes 19-21).

Algorithme 12 : Pursuit($G_n, p, s, cutoff$)

```
1  Let τ an empty trajectory
2  if G_n is empty then Update(G_n, VisibleCriticalPointsFrom(p) ∪{p})
3  while ∃ p ∈ G_n | p ∈ UnexploredCriticalPoints(G_n) do
4      τ_e ← Explore(p, G_n, cutoff)
5      τ ← τ ⊕ τ_e
6      UpdatePursuitState(s, τ_e, G_n)
7      if ∀ p ∈ s, p is labeled Clear then return τ
8      else
9          G_p ← ExpandPursuitGraph(G_n, p, s)
10         if ∃s' ∈ G_p | ∀p ∈ s', p is labeled Clear then
11             τ_p ← PlanPursuitTrajectory(G_p, p)
12             τ ← τ ⊕ τ_p
13             p ← Execute(τ_p)
14             UpdatePursuitState(s, τ_p, G_n)
15         else
16             choose a critical point p' to split the environment
17             components ← SplitEnvironment(G_n, p')
18             Deliberation(p')
19             forall G'_n ∈ components do
20                 τ_p ← Pursuit(G'_n, p, s, cutoff)
21                 τ ← τ ⊕ τ_p
22 return τ
```

7.4 Coopération et protocole de délégation

La coopération entre les poursuivants est réalisée grâce au protocole de délégation (cf. figure 7.5). Comme évoqué précédemment lorsqu'un poursuivant ne parvient pas à nettoyer seul une zone de l'environnement, il est dans l'obligation de coopérer. Cette coopération s'effectue en positionnant un poursuivant de manière stationnaire dans l'environnement afin de le découper en zones qui peuvent être surveillées par un seul poursuivant et éviter la recontamination des zones précédemment nettoyées. Lorsqu'une zone est nettoyée le poursuivant affecté à la surveillance ainsi que le poursuivant affecté au découpage de l'environnement peuvent être réalloués au nettoyage d'une autre zone. L'objectif du protocole de coopération est de minimiser le nombre de poursuivants nécessaires. Le protocole s'appuie sur le principe de moindre engagement : un poursuivant est ajouté si est seulement si aucun poursuivant déjà présent ne peut être réalloué à la nouvelle tâche. Le protocole s'appuie sur différents rôles que peuvent jouer les poursuivants pendant le processus de surveillance. Le rôle des poursuivants peut évoluer au cours de l'exploration de l'environnement. On distingue notamment les rôles suivants :

— *Le rôle d'explorateur.* Les explorateurs explorent une partie de l'environnement qui leur est affecté pour en construire une représentation ;

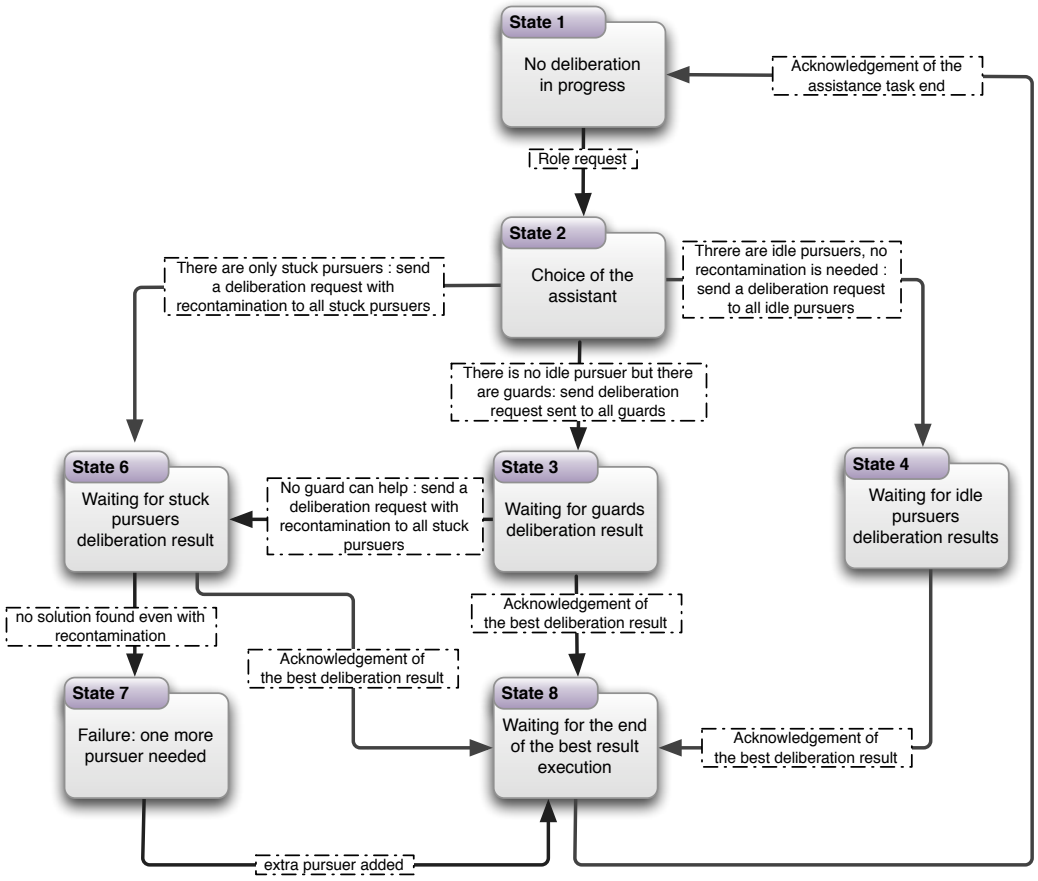


FIGURE 7.5 – Protocole de délibération entre poursuivants

— *Le rôle de garde.* Les gardes restent stationnaires. Ils se positionnent en un point stratégique de l'environnement pour le découper en zones qui peuvent être surveillées plus simples et empêcher les recontamination de l'environnement ;
— *Le rôle de poursuivants libres.* Les poursuivants libres ont terminé la tâche de surveillance qui leur avait été affectée. Ils peuvent être réalloués à un autre rôle.
— *Le rôle de poursuivants bloqués.* Les poursuivants bloqués ne peuvent progresser sans l'aide d'un autre poursuivant. Ils ne peuvent se déplacer sans risquer une recontamination de l'environnement.

La première étape du protocole de coopération consiste à collecter le rôle des différents poursuivants (cf. figure 7.5, état 1). Cette première étape du protocole exécutée par un poursuivant bloqué permet d'estimer la capacité du système à répondre à sa demande d'assistance. Lors que tous les rôles ont été collectés (cf. figure 7.5, état 2), trois cas doivent être envisagés :

Cas 1. Si des poursuivants libres sont présents, le poursuivant bloqué leur envoie une demande d'assistance en indiquant le point critique où il souhaiterait qu'un garde se positionne (cf. figure 7.5, état 4). Tous les poursuivants libres lui répondent alors en indiquant le coût pour lui venir en aide (cf. figure 7.5, état 8), i.e., la distance à parcourir pour que le poursuivant libre vienne se positionner depuis sa position actuelle à la position stationnaire nécessaire au découpage de l'environnement choisi par le poursuivant

bloqué. À la réception de toutes les réponses, le poursuivant bloqué demande au poursuivant libre ayant fait la meilleure proposition de venir se positionner à la position stratégique choisie. Lors ce que le poursuivant libre arrive à cette position, il informe le poursuivant bloqué qu'il peut poursuivre sa surveillance sans risque de recontamination et change de rôle. Il joue alors le rôle de garde. Finalement, tous les poursuivants repassent dans l'état 1 du protocole de coopération.

Cas 2. Si aucun poursuivant n'est libre, mais que des poursuivants jouant le rôle de gardes sont présents dans l'environnement, le poursuivant bloqué leur envoie les points critiques qu'il souhaiterait voir surveillés et l'état global de la surveillance (cf. figure 7.5, état 3). L'état global est calculé en fusionnant les états de surveillance de chaque poursuivant. À partir de ces informations, chaque garde est en mesure de calculer s'il peut se déplacer pour venir en aide au poursuivant bloqué tout en assurant son rôle initial de garde et en évitant les recontamination de l'environnement. Deux cas doivent être considérés :

Un garde s'aperçoit que les points critiques dont il avait la surveillance sont tous nettoyés. Par conséquent, sa tâche de garde n'est plus nécessaire. Il redevient alors un poursuivant libre. Il peut alors comme précédemment calculer une trajectoire pour venir en aide au poursuivant bloqué en se déplaçant à la position demandée sans risque de recontamination de l'environnement.

Un garde s'aperçoit qu'au moins un point critique dont il avait la surveillance est encore contaminé. Il ne peut donc pas se déplacer impunément sans risquer une contamination. Dans ce cas, le garde cherche un chemin de sa position actuelle à la position demandée par le poursuivant bloqué tel que les points contaminés restent visible à tout instant. Si un tel chemin existe alors le garde peut venir en aide au poursuivant bloqué tout en assurant sa tâche première de garde.

Quelque soit le cas (2.1) ou (2.2), les gardes envoient le coût de leur solution et le poursuivant bloqué choisit la meilleure proposition (cf. figure 7.5, état 8). Finalement, tous les poursuivants retournent dans l'état 1.

Cas 3. Si aucun des deux précédents cas n'aboutit, cela signifie qu'aucun poursuivant n'est disponible pour porter assistance au poursuivant bloqué sans recontaminer une partie de l'environnement précédemment nettoyée. Dans ce cas, chaque poursuivant évalue le coût pour venir en aide en évaluant le nombre de points critiques recontaminés par l'abandon de leur position et en estimant la distance à parcourir pour atteindre la position demandée par le poursuivant bloqué. Le second critère est utilisé pour départager deux solutions identiques sur la base du nombre de points critiques recontaminés. Comme précédemment, le poursuivant bloqué attend les réponses de l'ensemble de ses équipiers avant de choisir la meilleure proposition et de poursuivre son exploration. Notons que le poursuivant bloqué garde en mémoire les zones recontaminées pour éviter des boucles de recontamination. Si une telle boucle est détectée, la procédure échoue (cf. figure 7.5, état 7). Cela signifie que le nombre de poursuivants nécessaires pour nettoyer l'environnement n'est pas atteint. Un nouveau poursuivant est ajouté dans l'environnement en tant que garde à la position demandée par le poursuivant bloqué qui peut alors reprendre sa ronde.

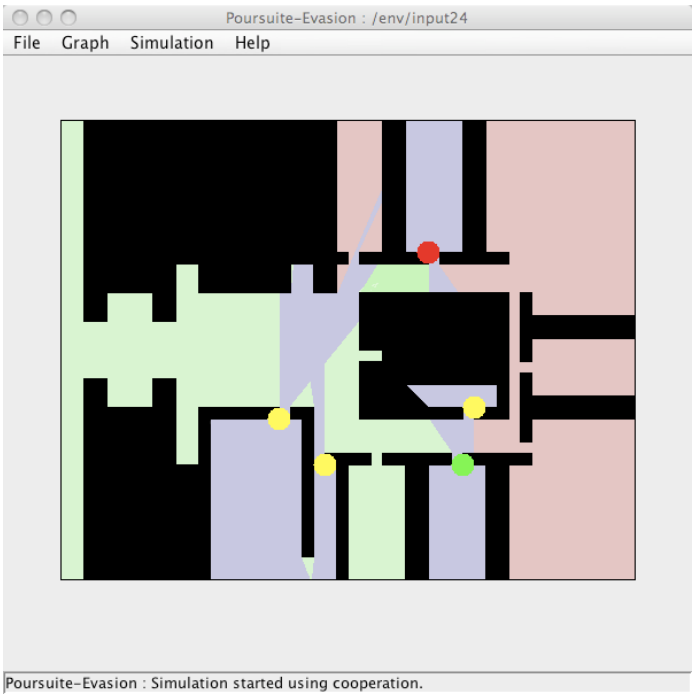


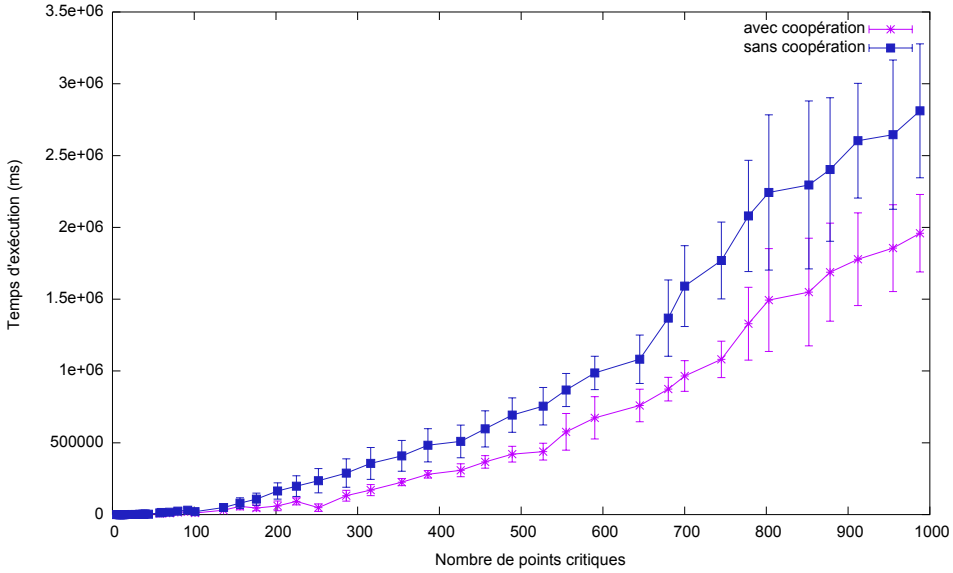
FIGURE 7.6 – Simulateur de poursuite-évasion : les zones bleues représentent la vision des poursuivants ; la zone verte la partie de l'environnement nettoyée ; la partie rouge la partie contaminée ; les poursuivants rouges explorent l'environnement ; les poursuivants jaunes délibèrent et les poursuivants verts sont des gardes.

7.5 Expérimentation et évaluation

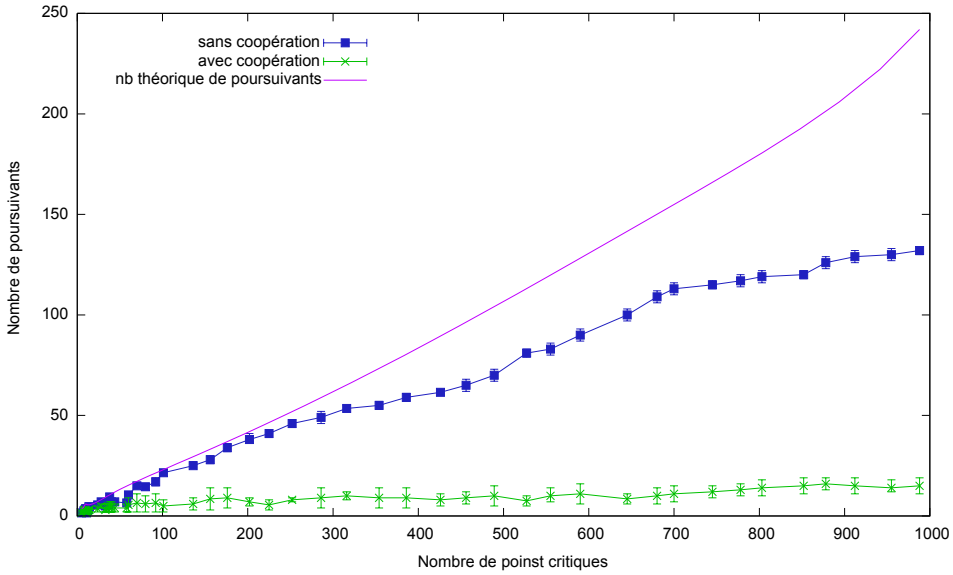
Afin d'évaluer la pertinence de notre approche, nous avons développé un simulateur (cf. figure 7.6). Ce simulateur a été implémenté avec le langage Java en s'appuyant sur les bibliothèques : JGraph [2] pour la gestion des graphes de navigation et de poursuite ; JST Topology Suite [3] pour les calculs géométriques et sur CoRe-DMS [4], que nous avons développé spécifiquement pour gérer les aspects multi-agents et les échanges asynchrones de messages entre poursuivants.

Notre jeu de tests se compose de 60 environnements générés de manière aléatoire. Les environnements ont été classés par complexité croissante, i.e., en fonction de leur nombre de points critiques. L'environnement le plus simple contient seulement 5 points critiques et le plus compliqué 975. Le point de départ choisi pour effectuer la surveillance peut avoir une influence sur le découpage en zone de l'environnement par les poursuivants bloqués et par conséquent sur le nombre de poursuivants nécessaires. Afin d'éliminer ce biais, nous avons effectué pour chaque environnement une simulation en choisissant chaque point critique de l'environnement comme point de départ de la simulation. Les résultats donnés sont donc des moyennes. Les résultats avec ou sans coopération sont donnés à la figure 7.7(a) pour ce qui est du temps d'exécution et à la figure 7.7(b) pour ce qui est du nombre de poursuivants. Les barres d'erreur indiquent la variation observée en fonction du point critique choisi comme point de départ de la simulation.

2. https://www.jgraph.com
3. https://github.com/locationtech/jts
4. http://core.imag.fr



(a) Évolution du temps d'exécution en fonction de la complexité de l'environnement



(b) Évolution du nombre de poursuivants nécessaires en fonction de la complexité de l'environnement

Coopération versus non-coopération

L'approche avec coopération surpasse celle sans coopération en temps, mais aussi en nombre de poursuivants nécessaires. On peut ainsi espérer un gain en temps de plus de 25% entre l'approche coopérative et l'approche non-coopérative. En ce qui concerne le nombre de poursuivants cette fois, le gain est beaucoup plus important. Il est de 2 pour les petits environnements et de plus de 10 pour les environnements testés plus complexes.

Influence du point de départ de la simulation

D'une manière générale, l'influence du point de départ choisi pour la simulation est plutôt marginale, quelle que soit la stratégie, avec ou sans coopération.

	Avec coopération	Sans coopération
Moyenne	0,20	0,18
Max	0,29	0,32
Min	0,12	0,05
Variation	0,05	0,08

TABLE 7.1 – Variation du temps d'exécution

	Avec coopération	Sans coopération
Moyenne	0,27	0,096
Max	0,47	0,37
Min	0,15	0
Variation	0,064	0,087

TABLE 7.2 – Variations du nombre de poursuivants

La variation n'excède jamais 29% avec coopération et 32% sans coopération au regard du temps d'exécution et de 47% et 37% au regard du nombre de poursuivants nécessaires (cf. table 7.1 et table 7.2). Dans tous les cas, le protocole de coopération a tendance à diminuer l'influence du point de départ choisi pour la ronde des poursuivants.

7.6 Conclusion

Dans ce chapitre, nous avons présenté une approche originale pour résoudre le problème de poursuite-évasion dans des environnements labyrinthiques 2D inconnus. Nous avons supposé que les poursuivants possédaient une vision omnidirectionnelle non limitée et qu'ils agissaient de manière collaborative. La comparaison de l'approche avec et sans coopération a montré qu'un mécanisme simple de coopération comme celui présenté pouvait améliorer de manière significative le temps d'exécution, mais également le nombre de robots nécessaires à la résolution du problème.

Ce travail ouvre la voie à plusieurs extensions. En particulier, il serait intéressant de considérer des cas plus réalistes où les perceptions des robots seraient restreintes (angle et distances de perception) comme certains travaux l'on fait dans un cadre mono-robot. Il serait également intéressant, contrairement à beaucoup de travaux de la littérature, d'essayer non plus de minimiser le nombre de poursuivants pour résoudre le problème, mais plutôt de chercher à minimiser le temps ainsi que la trajectoire globale des poursuivants en ce qui concerne la distance. Il existe peu ou pas de résultats théoriques concernant ces deux aspects à notre connaissance. Pour terminer, nous pouvons évoquer une dernière piste de recherche. Très peu de travaux se sont intéressés au problème de poursuite-évasion dans des environnements en trois dimensions. Quelques résultats théoriques existent, mais il n'existe pas pour l'instant d'algorithmes efficaces.

Troisième partie

Planification et heuristiques

Le domaine de la planification automatique a réalisé des avancées majeures en ce qui concerne les performances ces dernières années (Bryce et Kambhampati, 2007 ; Liang, 2012). Ceci s'explique en partie grâce au développement et à la découverte de fonctions heuristiques, ou plus simplement d'heuristiques, de plus en plus informatives pour guider l'exploration d'espaces de recherche de plus en plus grands. Malgré ces avancées, la performance des algorithmes de planification reste critique lorsque l'on cherche à aborder des problèmes réels.

La conception de nouvelles heuristiques repose sur le *principe de relaxation*. Ce principe consiste à résoudre un problème plus simple en ignorant volontairement certaines de ces contraintes. L'objectif est d'obtenir un problème d'une classe de complexité inférieure, facilement soluble, dont le résultat peut fournir une bonne estimation du coût de la résolution du problème non relaxé initial. La littérature distingue principalement cinq grandes familles d'heuristiques (Helmert et Domshlak, 2009) :

1. *Les heuristiques fondées sur le calcul du chemin critique* (Haslum, Bonet et Geffner, 2005 ; Haslum et Geffner, 2000) estiment la distance au but en calculant une borne inférieure pour atteindre un sous-ensemble des propositions du but d'une taille prédéfinie. Plus l'ensemble de propositions est grand, plus l'heuristique est informative, mais également plus elle est coûteuse à calculer.
2. *Les heuristiques fondées sur la relaxation des effets négatifs* (Domshlak, Hoffmann et Katz, 2015 ; Hoffmann et Nebel, 2001 ; Nguyen, Kambhampati et Nigenda, 2002) estiment le coût pour atteindre le but en calculant la solution au problème de planification simplifié dans lequel les actions n'ont pas d'effets négatifs. Sous l'hypothèse classique que le nombre d'objets du monde est fini, l'espace de recherche du problème simplifié sature très rapidement ce qui permet de calculer très rapidement une solution optimale au problème simplifié.
3. *Les heuristiques fondées sur l'abstraction* (Edelkamp, 2001 ; Helmert et al., 2014 ; Katz et Domshlak, 2008) essaient de factoriser les états de l'espace de recherche en fonction de leurs propriétés afin de le réduire. Lorsque les mécanismes d'abstraction réduisent suffisamment l'espace de recherche, il est possible de trouver rapidement une solution optimale au problème factorisé. La solution est utilisée comme estimation pour guider la recherche du problème non factorisé.
4. *Les heuristiques fondées sur les landmarks* (Hoffmann, Porteous et Sebastia, 2004 ; Richter et Westphal, 2008) se fondent sur l'observation que certaines propositions sont vraies pour tous les plans solutions d'un problème donné. Déterminer ces points de passage obligés réduit la complexité du problème initial en décomposant le problème initial en un ensemble de sous problèmes plus ou moins indépendants.

5. *Les heuristiques fondées sur l'ajout de connaissances* (GEORGIEVSKI et MARCO, 2015 ; JONSSON, 2007) se fondent sur l'ajout ou l'apprentissage de connaissances spécifiques au domaine de planification pour améliorer les performances et guider la recherche d'un plan solution.

Dans cette partie, nous allons présenter deux de nos contributions à la planification et à la recherche heuristiques qui s'inscrivent dans la dernière grande famille d'heuristique. Le chapitre 8 présente une technique pour apprendre de manière incrémentale des macro-actions, i.e., des séquences d'actions fréquemment utilisées. Nous présenterons également comment exploiter ces macro-actions pour accélérer la recherche de nouveaux plans solutions. Finalement, le chapitre 9 présente une technique pour simplifier la représentation des problèmes de planification HTN (Hierarchical Task Networks) et en accélérer leur résolution.

Chapitre 8

Apprentissage de macro-actions pour la planification

Peu de techniques de planification sont capables de tirer parti des précédents problèmes résolus pour accélérer la recherche d'un nouveau problème. Pourtant, intuitivement, un système capable d'exploiter son expérience devrait être capable d'obtenir de meilleures performances. Prenons le cas d'un aspirateur qui doit jour après jour nettoyer votre intérieur. Chaque jour, la tâche à réaliser est légèrement différente, e.g., une porte habituellement fermée est ouverte, un fauteuil a été déplacé, une paire de chaussettes a été négligemment laissée sur le sol, etc. Toutefois, une grande partie de la trajectoire de l'aspirateur reste inchangée. La problématique est donc la suivante : comment un système autonome et *a fortiori* le planificateur qu'il intègre est-il capable d'apprendre de ses précédentes décisions des routines pour capitaliser à partir de son expérience ?

Pour répondre à cette question, l'approche proposée[1] dans ce chapitre (Dulac et al., 2013) consiste à apprendre ce que nous appelons des *macros*. Les macros modélisent les routines du système. Les macros une fois apprises peuvent être réinjectées directement dans le domaine de planification pour tirer parti des précédentes recherches. Une macro est une séquence choisie d'actions à un instant donné qui s'applique comme une action classique. Les macros permettent de faire des sauts dans l'espace de recherche en construisant des états à la fois profonds et prometteurs pour atteindre un état but. Soigneusement choisies, les macros peuvent améliorer de manière significative les performances en réduisant la profondeur de l'espace de recherche. Le principal inconvénient des macros est d'augmenter le coefficient de branchement de l'espace de recherche. Par conséquent, l'utilisation de macros soulève un *problème d'utilité*. Il faut trouver un compromis entre le bénéfice escompté par l'ajout de macros et le surcoût induit par l'augmentation du coefficient de branchement.

L'approche décrite dans ce chapitre gère au mieux le problème de l'utilité. Elle fonctionne avec n'importe quels planificateur ou domaine de planification. Elle n'exploite aucune structure de données spécifiques aux planificateurs et aucune connaissance spécifique du domaine. Étant donné un planificateur, un domaine et un ensemble de problèmes, notre approche effectue une analyse statistique fondée sur les n-grammes pour générer les macros pertinentes. Les macros ainsi produites sont réinjectées après une étape de généralisation et de filtrage dans la description du domaine de planification pour améliorer les recherches suivantes. La technique des n-grammes est une technique classique utilisée en traitement du langage naturel pour la reconnaissance ou la traduction automatiques pour calculer la probabilité d'occurrence des mots en fonction d'un historique observé.

1. L'approche a été développée dans le cadre du stage de master 2 recherche d'Adrien Dulac.

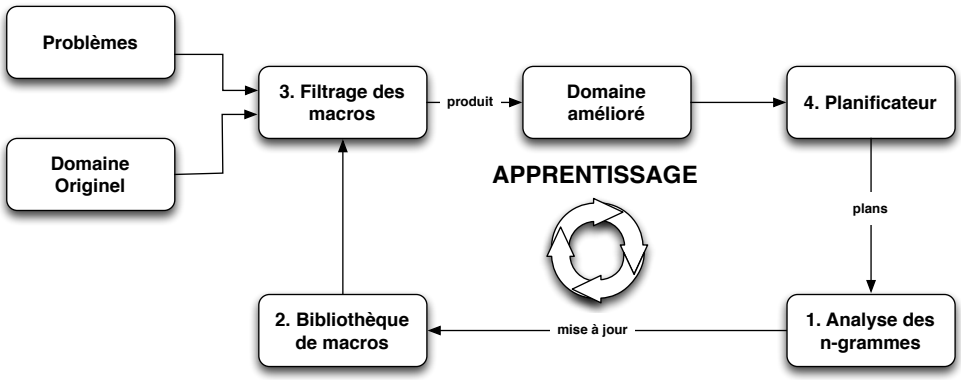


FIGURE 8.1 – Boucle d'apprentissage et de planification avec macro-actions

Le reste du chapitre est organisé de la manière suivante. Une première section présente succinctement les travaux relatifs à l'utilisation de macros dans le domaine de la planification. Une seconde section introduit le principe et les concepts de notre approche. Ma troisième détaille les différentes étapes de notre contribution qui conduisent à l'ajout des macros dans le domaine modifié. Nous terminons par une évaluation empirique de notre contribution sur des domaines et des problèmes issus des compétitions internationales de planification pour montrer sa pertinence.

8.1 État de l'art : Planifier avec des macro-actions

La littérature distingue deux familles d'approches : les approches hors ligne et les approches en ligne.

8.1.1 Les approches hors ligne

La plupart des techniques développées en planification automatique pour apprendre à utiliser des macros, et ce depuis les travaux précurseurs de (FIKES, HART et NILSSON, 1972; IBA, 1989; KORF, 1985), reposent sur une approche hors ligne. Les macros sont générées puis testées sur un ensemble de problèmes relativement simples pour déterminer leur pertinence à résoudre de nouveaux problèmes.

On retrouve cette approche dans des travaux récents tels que ceux proposés par (BOTEA et al., 2005) et le planificateur Macro-FF. Ce planificateur fonctionne en quatre temps : (1) une analyse de la structure du domaine de planification pour identifier des connaissances implicites; (2) la génération des macros à partir des plans solutions trouvés sur un ensemble restreint de problèmes de planification servant d'ensemble de tests (3) la sélection des macros les plus prometteuses en fonction de règles heuristiques et (4) finalement, l'utilisation des macros choisies pour résoudre de nouveaux problèmes.

L'approche de (NEWTON et al., 2007) repose sur les mêmes principes. L'originalité de ces travaux réside dans le fait que la sélection des macros est réalisée par un algorithme d'apprentissage génétique. Les macros sont choisies de manière aléatoire à partir des plans solutions générés à partir d'un ensemble de problèmes de tests pour constituer la soupe initiale. Afin d'explorer uniquement les macros appartenant aux plans solutions générés, les opérateurs génétiques sont limités à des opérations portant sur des actions adjacentes des plans solutions : suppression ou ajout d'une action en début ou en fin de macro, découpage d'une macro en deux sous-macros,

ou encore ajout aléatoire d'une nouvelle macro. L'évaluation est réalisée en calculant le temps moyen pour résoudre l'ensemble des problèmes de test avec ou sans macros. La principale limitation de cette approche est le temps de convergence vers l'ensemble des macros les plus adaptées.

Une autre approche proposée par (BOTEA, MÜLLER et SCHAEFFER, 2005) élabore les macros à partir d'un ensemble de problèmes de tests en construisant pour chaque problème une structure de données appelée *un graphe solution*. Les nœuds du graphe sont les actions du plan solution et les arcs les liens de causalité entre les actions. Chaque sous-graphe du graphe solution est considéré comme une macro. Chaque macro est ensuite classée en fonction de règles et filtrée dynamiquement lors de la résolution d'un nouveau problème en utilisant l'heuristique du planificateur FastForward (HOFFMANN et NEBEL, 2001). Cette approche a été généralisée à des macros itératives, i.e., des macros de macros, par (BOTEA, MÜLLER et SCHAEFFER, 2007).

Une dernière technique que nous devons mentionner est celle proposée par (CHRPA, VALLATI et MCCLUSKEY, 2014, 2015). L'idée de l'approche est de guider l'extraction des macros en calculant les relations entre les opérateurs de planification, l'état initial et le but du problème. Ces relations appelées *outer entanglement* permettent de filtrer les macros inutiles pour un problème donné, limitant le coefficient de branchement de l'espace de recherche et améliorant ainsi les performances du processus de planification.

8.1.2 Les approches en ligne

Par opposition, les approches en ligne ne nécessitent aucune phase préalable de tests sur un ensemble de problèmes prédéfinis. Ces techniques sont moins nombreuses. On peut citer toutefois les travaux de (COLES et SMITH, 2007) et du planificateur Marvin. Marvin mémorise les macros pour sortir plus rapidement des plateaux de l'espace de recherche engendrés notamment par l'algorithme de recherche *Enforced Hill Climbing* utilisé par FastForward (HOFFMANN et NEBEL, 2001). La principale limitation de cette approche réside dans le fait qu'elle est spécifique à l'algorithme de recherche utilisé. Pour terminer, nous pouvons évoquer la technique proposée par (ASAI et FUKUNAGA, 2015; JONSSON, 2009). Cette technique décompose un problème initial en un ensemble de sous-problèmes. Chaque solution partielle à un sous-problème est alors considérée comme une macro qui peut être utilisée pour la résolution du problème initial.

En conclusion, nous avons les approches hors ligne qui nécessitent une phase de test pour apprendre les macros sur un ensemble prédéfini de problèmes qui ne reflètent pas forcément l'expérience du système, et les approches en ligne qui sont pour la plupart dépendantes du planificateur utilisé. L'approche que nous proposons essaie de répondre à ces deux critiques.

8.2 Principes et concepts de notre approche

8.2.1 Des n-grammes aux macros

Notre contribution repose sur la technique des n-grammes. Cette technique a été très utilisée dans le domaine du traitement automatique de la langue naturelle pour la reconnaissance automatique de la langue naturelle. Cette technique repose sur l'hypothèse de Markov : l'occurrence d'un événement ne dépend que de l'état précédant l'événement. Dans le contexte du traitement automatique des langues, les n-grammes représentent des sous-séquences de n mots extraits d'un corpus. Les

Algorithme 13 : GeneralizeMacro($\langle a_1, \ldots, a_n \rangle$)

Let o an empty operator
foreach a_i *in* $\langle a_1, \ldots, a_n \rangle$ **do** $o \leftarrow$ `Merge`(o, a_i)
Replace each occurence of a constant c in o by a parameter p
return o

Algorithme 14 : Merge(a_1, a_2)

$m \leftarrow a_1$
foreach *proposition* $p \in$ *precond*(a_2) **do**
  if $p \notin$ *effects*$^+(m) \cup$ *precond*(m) **then** precond$(m) \leftarrow$ precond$(m) \cup \{p\}$
foreach *proposition* $p \in$ *effect*$^-(a_2)$ **do**
  if $p \in$ *effect*$^+(m)$ **then** effects$^+(m) \leftarrow$ effects$^+(m) \backslash \{p\}$
  else effects$^-(m) \leftarrow$ effects$^-(m) \cup \{\}$
foreach *proposition* $p \in$ *effects*$^+(a_2)$ **do**
  if $p \in$ *effects*$^-(m)$ **then** effects$^-(m) \leftarrow$ effects$^-(m) \backslash \{p\}$
  effects$^+(m) \leftarrow$ effects$^+(m) \cup \{p\}$
return m

n-grammes sont utilisés pour calculer la probabilité de l'occurrence du n-ième mot considérant les $n-1$ prédécesseurs. La probabilité obtenue peut être alors utilisée pour déterminer les séquences de mots ou les phrases les plus probables.

Dans notre approche, l'ensemble des plans solutions correspondants aux problèmes déjà résolus $P = \{\pi_1, \ldots, \pi_k\}$ constitue le corpus d'apprentissage. Un n-gramme $s = \langle a_1, \ldots, a_n \rangle$ est une séquence d'actions de taille n à laquelle on associe une probabilité d'occurrence dans le corpus, ce qui permet de les classer du plus probable au moins probable. La procédure qui généralise les n-grammes en macro-opérateurs est donnée par l'algorithme 13. La procédure consiste à fusionner incrémentalement les actions du n-gramme en utilisant la procédure de fusion (cf. Algo. 14) pour construire une macro-action, i.e., une macro complètement instanciée, puis à la généraliser en remplaçant chaque occurrence d'une constante par une variable. Les variables introduites deviennent les paramètres du macro-opérateur généralisé. Notons que plusieurs n-grammes peuvent conduire à la généralisation d'un même macro-opérateur à cause de l'étape de généralisation. Par ailleurs, nous dirons qu'un macro-opérateur est d'ordre n s'il a été produit par généralisation d'un n-gramme de taille n. En pratique seuls les macro-opérateurs d'ordre 2, i.e., regroupant au moins deux actions, sont intéressants.

8.2.2 Le problème de l'utilité des macro-opérateurs

Le problème de l'utilité, i.e., du choix des macros pertinentes à capitaliser au travers des multiples résolutions, est crucial. En effet, si l'on ajoute trop de macro-opérateurs au domaine le coefficient de branchement croît inutilement, perdant ainsi l'avantage lié à leur usage pour guider la recherche d'un plan solution. Par ailleurs, l'ordre des macro-opérateurs peut également avoir une influence. Intuitivement, l'ajout de macro-opérateurs d'ordre élevé semble une bonne idée dans la mesure

où ces macro-opérateurs permettent de faire de grands sauts dans l'espace de recherche. En pratique, les macro-opérateurs d'ordre élevé, même s'ils peuvent produire un gain important, sont très rares et donc peu utiles. De plus, l'ajout de macro-opérateurs d'ordre trop élevé n'est pas sans poser de problèmes pratiques au processus d'instanciation réalisé par la plupart des planificateurs modernes. L'instanciation d'un opérateur en actions est en effet exponentielle en fonction de son nombre de paramètres, et son nombre de paramètres est d'autant plus élevé que l'ordre du macro-opérateur est élevé. En conséquence, le critère d'utilité que nous devons utiliser doit permettre (1) de sélectionner les macro-opérateurs en fonction de leur potentialité à être réutilisés en fonction du corpus de plans représentant l'expérience du système, mais aussi (2) de déterminer l'ordre des macro-opérateurs le plus adapté.

La première partie du critère est évaluée en calculant ce que nous appelons la couverture d'un macro-opérateur d'ordre n. La couverture d'un macro-opérateur m d'ordre n noté $C(n,p)$ mesure le nombre minimum de macro-opérateurs nécessaires pour couvrir le pourcentage p des occurrences des n-grammes du corpus de plans P représentant l'expérience du système :

$$C(n,p) = \text{argmin}_{s \in S(n,p)} |s| \tag{8.1}$$

tel que :

$$S(n,p) = \{s \in 2^M | \sum_{m \in s} \mathbb{P}(m) \geq p\} \tag{8.2}$$

Dans l'équation 8.2, M représente l'ensemble des macro-opérateurs m d'ordre n et $\mathbb{P}(m)$ définit la probabilité de vraisemblance de m sur le corpus P. $\mathbb{P}(m)$ est défini comme suit :

$$\mathbb{P}(m) = \sum_{n\text{-gram}} \mathbb{P}(a_n \mid a_{n-1}, .., a_0) = \frac{w(m)}{\sum_{m' \in M} w(m')} \tag{8.3}$$

où $w(m)$ est le nombre d'occurrences de n-grammes produisant m à partir de P en utilisant l'algorithme 13.

Maintenant que nous avons un critère pour choisir les macro-opérateurs d'ordre n ayant la meilleure couverture de notre ensemble de plans, il reste à déterminer l'ordre n le plus adapté. Comme précédemment évoqué, le choix de l'ordre doit être (1) un compromis entre longueur et couverture, mais également (2) tenir compte de l'explosion combinatoire induite par leur instanciation.

En ce qui concerne le premier point, nous définissons l'ordre idéal en fonction de la couverture n^c comme le compromis entre la longueur des macro-opérateurs et la concentration des n-grammes les plus probables dans une distribution de n-grammes. Cette concentration est évaluée en comparant un ratio de généralisation $G(n)$ représentant le nombre moyen de n-grammes généralisés dans un macro-opérateur et sa couverture [2] comme suit :

$$n^c = \begin{cases} n \text{ tel que } G(n) = C(n,p) \\ \text{argmax}_n(G(n)) \qquad \text{sinon.} \end{cases} \tag{8.4}$$

En ce qui concerne le second point, nous définissons l'ordre idéal en termes de mémoire n^m comme suit :

$$Q = \log_2(\Delta(n^m)) \tag{8.5}$$

2. Les deux valeurs sont homogènes. Elles représentent des quantités de macros-opérateurs

où Q représente la quantité de mémoire disponible et $\Delta(n)$ une borne supérieure [3].

Pour conclure, l'ordre idéal n^* est un compromis entre n^c et n^m :

$$n^* = \min(n^c, n^m), \tag{8.6}$$

et l'ensemble M^* des macro-opérateurs les plus utiles est défini par :

$$M^* = \{m \in S(n^*, p) \mid \forall m' \in S(n^*, p),\ |m| < |m'|\} \tag{8.7}$$

8.3 Les grandes étapes de l'approche

L'approche proposée est une approche en ligne qui ne nécessite aucun entraînement préalable sur un ensemble de problèmes de tests prédéfinis. Elle est domaine et planificateur indépendante. Elle n'exploite que les plans solutions précédemment trouvés pour déduire les macros les plus pertinentes pour résoudre un nouveau problème. Nous nous plaçons dans le cadre de la planification STRIPS. Notre approche s'articule autour de quatre étapes (cf. figure 8.1) :

1. *Une étape d'analyse* statique qui consiste à calculer les n-grammes, à partir de l'ensemble des plans solutions précédemment trouvés représentant l'expérience du système ;
2. *Une étape de génération* qui consiste à construire une bibliothèque de macro-opérateurs à partir de l'étape d'analyse respectant le critère d'utilité précédemment donné sans tenir compte du problème particulier à résoudre ;
3. *Une étape de filtrage* qui consiste à sélectionner dynamiquement les macros les plus pertinentes pour un problème spécifique ;
4. *Une étape de planification* qui consiste à planifier en utilisant un domaine de planification amélioré contenant les macros obtenues à l'étape de filtrage.

8.3.1 L'étape d'analyse

L'hypothèse sous-jacente à l'utilisation de macros est que des routines sont présentes dans les problèmes déjà résolus et que ces routines peuvent être utilisées pour améliorer les performances du processus de résolution de nouveaux problèmes. Par conséquent, chaque nouveau plan calculé doit être stocké pour être dans un second temps analysé. La question qui se pose alors est de savoir quelle propriété ou quelle taille doit avoir le corpus de plans solutions stockés avant de pouvoir effectuer une analyse pertinente pour déterminer les macro-opérateurs utiles ? Dans l'idéal il serait souhaitable que la distribution des n-grammes extraits du corpus de plans solutions soit stable avant de débuter l'analyse. Toutefois, en pratique, nous avons observé que le nombre de n-grammes différents augmentait avec le nombre de plans stockés en suivant une loi de Heaps (HEAPS, 1978). Quel que soit l'ordre des n-grammes, le nombre de n-grammes dépend du nombre d'opérateurs de planification, de leur nombre de paramètres et du nombre de plans solutions du corpus. Par conséquent, il n'est pas possible d'utiliser le nombre de n-grammes différents pour déterminer quand effectuer l'analyse préliminaire du

3. Une borne supérieure pour un macro-opérateur peut être calculée avec la formule suivante : $|C|^k$ où C représente l'ensemble des constantes du problème et k le nombre de paramètres du macro-opérateur sur le nombre d'actions résultant de l'instanciation de tous les macro-opérateurs d'ordre n

corpus de plans nécessaire à la génération des macro-opérateurs. Toutefois, nos observations (cf. §8.4) ont montré que le nombre de n-grammes les plus fréquents devenait stable au fil du temps. Nous utilisons donc ce critère pour déclencher l'analyse et le passage à l'étape 2 de génération des nouveaux macro-opérateurs.

Finalement, les informations extraites du corpus de plans solutions sont de deux types :

1. *Une estimation du coefficient de branchement du domaine* calculé comme le rapport entre le nombre d'actions et le nombre de propositions des problèmes instanciés précédemment résolus ;
2. *Une distribution des n-grammes ainsi que leur nombre d'occurrences* pour chaque ordre dans le corpus de plans solutions.

8.3.2 La génération des macro-opérateurs

La génération de la bibliothèque de macro-opérateurs indépendamment d'un problème donné est réalisée en deux étapes : (1) une étape de généralisation des macro-opérateurs qui s'appuie sur le critère d'utilité précédemment introduit et la procédure de généralisation des macro-opérateurs (cf. §8.2) et (2) une étape de spécialisation permettant de réduire le nombre d'actions potentiellement induites par l'instanciation des macro-opérateurs.

Étape 1. (Généralisation des macro-opérateurs)

La génération des macro-opérateurs à partir des n-grammes est présentée §8.2.1 : l'algorithme 13 généralise les macros-actions en macro-opérateurs et l'algorithme 14 fusionne les différentes actions composant un n-gramme pour construire les macro-actions. Étant donné que le coefficient de branchement croît de manière exponentielle avec le nombre de paramètres d'un macro-opérateur, le processus de généralisation induit une augmentation importante de celui-ci. Il est possible d'obtenir une estimation du surcoût engendré en comparant le coefficient de branchement sans macro-opérateur déterminé à l'étape d'analyse avec le coefficient de branchement avec les macro-opérateurs. Nous avons fait le choix de fixer arbitrairement une limite à ce surcoût. Lorsque cette limite est dépassée, une étape de spécialisation est réalisée pour maîtriser l'augmentation du coefficient de branchement.

Étape 2. (La spécialisation des macro-opérateurs)

La spécialisation d'un macro-opérateur consiste à diminuer son nombre de paramètres en remplaçant un ou plusieurs de ses paramètres par une constante. Le processus de spécialisation est un nouveau compromis pour traiter le problème d'utilité entre le nombre de macro-actions générées lors de la recherche d'un plan solution et donc l'augmentation du coefficient de branchement et l'utilité de conserver des macro-opérateurs *utiles* pour améliorer les performances de la résolution de nouveaux problèmes. La procédure de spécialisation (cf. algo. 15) se déroule en deux étapes :

Sous-étape 1. (Calcul des constantes les plus probables) Pour chaque macro-opérateur et pour chaque paramètre, nous calculons à partir du corpus de plans solutions la constante qui a conduit le plus souvent à cette généralisation.

Algorithme 15 : SpecializeMacro(m)

```
Let l an empty list
foreach parameter p of m do
    e ← (p, c) s.t. ∀c' ∈ C : ℙ(c | p) > ℙ(c' | p)
    l ← l ⊕ e
l ← Sort (p, c) in l wrt. ℙ(c | p)
foreach (p, c) ∈ l do
    if ℙ(c | p) ≥ p_inf or k > free_sup then
        Substitute p by c in m
        k ← k − 1
return m
```

Sous-étape 2. (Instanciation des paramètres) Les paramètres correspondant aux k constantes les plus probables sont instanciés. La limite est fixée par un seuil défini de manière arbitraire.

8.3.3 Le filtrage des macro-opérateurs

L'étape précédente produit une bibliothèque de macro-opérateurs de trop grande taille pour être utilisée directement par un planificateur. En outre, elle ne tient pas compte des spécificités des nouveaux problèmes à résoudre. Un double filtrage est donc nécessaire : le premier filtrage, statique, pour réduire le nombre de macro-opérateurs en ne conservant que les macro-opérateurs ayant le meilleur ordre ; le second filtrage, cette fois dynamique, pour ne retenir que les macro-opérateurs pertinents pour un problème donné.

Le filtrage statique

Le filtrage statique qui consiste à déterminer le meilleur ordre pour les macro-opérateurs s'appuie sur la notion de meilleur ordre défini en §8.2.2. C'est un compromis entre la longueur des macro-opérateurs et le gain escompté, la mémoire disponible et la probabilité qu'il soit réutilisé pour résoudre un nouveau problème. Le filtrage conduit donc à ne retenir que les macro-opérateurs d'ordre n^*.

Le filtrage dynamique

Le filtrage dynamique tient compte des spécificités du nouveau problème à résoudre. Il consiste à choisir parmi l'ensemble des macro-actions instanciées à partir des macro-opérateurs d'ordre n^* lesquelles sont les plus utiles. Une fois encore l'objectif est de réduire autant que possible le coefficient de branchement tout en ajoutant les macro-actions les plus utiles pour résoudre le nouveau problème.

Le filtrage dynamique peut être vu comme une heuristique. Il s'appuie sur une structure de données, appelée le *graphe de planification relaxé*, initialement développée et utilisée par le planificateur FastForward (HOFFMANN et NEBEL, 2001) et fondée sur le concept de graphe de planification dont nous avons déjà parlé au chapitre 5 (cf. définition 5.2). Un graphe de planification relaxé est une structure de données qui encode en temps polynomial l'espace de recherche d'une version simplifiée d'un problème de planification ou les actions n'ont pas d'effets négatifs. Le plan solution du problème simplifié, extrait du graphe en temps polynomial, est proche de

la solution du problème non simplifié et peut être utilisé comme une heuristique informative pour guider la recherche d'un plan solution. Notre technique de filtrage dynamique exploite la propriété de ce plan. La première étape du filtrage dynamique consiste donc à construite le graphe de planification relaxé pour le nouveau problème de planification à résoudre à partir de son état initial, de son but ainsi que l'ensemble des actions et des macro-actions. Une fois le plan extrait du graphe de planification relaxé ne sont considérées comme utiles que les macro-actions présentes dans le plan solution du problème simplifié. Seules ces macro-actions sont ajoutées au domaine pour effectuer la résolution du nouveau problème, cette fois non simplifié. Ce filtrage présente l'avantage de réduire de manière drastique le nombre de macro-actions finalement ajoutées au domaine tout en prenant en compte les spécificités des nouveaux problèmes à résoudre.

8.4 Évaluation et expérimentation

L'objectif de nos expérimentations est double : d'une part, montrer que les macros apprises améliorent de manière significative les performances en temps du processus de planification sans diminuer la qualité des plans solutions, et d'autre part montrer l'influence du choix de l'ordre des macro-opérateurs sur le temps de recherche.

8.4.1 Cadre expérimental

Nos expériences ont été réalisées sur 9 domaines issus des compétitions internationales de planification dans la catégorie apprentissage [4] : Blockworld, Ferry, Depot, Satellite, Mprime, Gripper, Grid, Parking et Barman. Pour chaque domaine, 10 000 problèmes ont été générés aléatoirement avec les générateurs de problèmes fournis par la compétition. La distribution des problèmes générés est donnée par le tableau 8.1. Les distributions utilisées sont celles utilisées dans les compétitions. Les 1 000 problèmes les plus simples sont retenus pour constituer le corpus nécessaire à l'apprentissage des macros. Nos expérimentations ont montré que ce nombre était suffisant pour obtenir une distribution stable des n-grammes. 30 problèmes parmi les plus difficiles sont choisis aléatoirement pour comparer les performances avec et sans ajouts de macros au domaine. Le planificateur de test utilisé pour réaliser nos expérimentations est un simple planificateur en chaînage avant utilisant l'algorithme A* disponible dans la bibliothèque PDDL4J [5].

Toutes les expérimentations ont été effectuées sur une machine équipée d'un processeur Intel Xeon cadencé à 2,4GHz. La mémoire allouée à chaque expérimentation est de 16Go. Le temps maximal alloué est de 2 500 secondes. Nous avons limité arbitrairement la taille des n-gramme à 7 et le nombre maximum de paramètres autorisés dans les macro-opérateurs à 4. De plus, nous avons limité à 10 le nombre de macro-opérateurs pouvant être ajoutés à un domaine. Finalement, nous avons fixé le pourcentage de couverture nécessaire au calcul du critère d'utilité à 50%.

4. Une description complète des domaines et des générateurs de problèmes est disponible : http://www.icaps-conference.org

5. https://github.com/pellierd/pddl4j

Blocksworld		**Ferry**		**Parking**	
Paramètres	*Valeurs*	*Paramètres*	*Valeurs*	*Paramètre*	*Valeurs*
blocks	[3-30]	cars	[1-200]	curbs	[2-8]
		locations	[1-8]	cars	[1-15]

Gripper		**Satellite**		**Depot**	
Paramètres	*Valeurs*	*Paramètres*	*Valeurs*	*Paramètres*	*Valeurs*
robots	[1-8]	satellites	[1-6]	depots	[1-5]
rooms	[1-8]	instruments	[1-2]	distributors	[1-3]
balls	[1-75]	modes	[1-8]	trucks	[1-4]
		x	[1-2]	pallets	[1-3]
		observation	[1-20]	hoists	[1-3]
				crates	[1-20]

Grid		**Mprime**		**Barman**	
Paramètres	*Valeurs*	*Paramètres*	*Valeurs*	*Paramètres*	*Valeurs*
x-scale	[3-8]	locations	[2-8]	cocktails	[1-30]
y-scale	[3-8]	fuel	[1-8]	ingredients	[1-13]
key+lock	[1-8]	spaces	[1-8]	shots	[1-30]
keys	[1-8]	vehicles	[1-8]		
locks	[1-8]	cargos	[1-8]		
goal proba.	[50-100]				

TABLE 8.1 – Distribution des problèmes générés

8.4.2 Analyse des performances

La figure 8.2 présente les résultats obtenus sans macro, avec des macro-opérateurs d'ordre 2 et macro-opérateurs d'ordre idéal n^*. Les résultats de la figure 8.2 sont résumés dans le tableau 8.2 en termes de gain relatif par rapport à notre planificateur de référence n'utilisant pas de macros. Pour chaque domaine et pour chaque domaine amélioré par l'ajout de macro-opérateurs, nous donnons le pourcentage moyen de gain en temps de recherche T ainsi qu'en longueur de plan Q : t et q représentent respectivement la dispersion du temps et de la longueur des plans.

Quel que soit le domaine, l'ajout de macro-opérateurs améliore les performances du planificateur de référence. En ce qui concerne le temps, le gain est important. Il est compris entre 14% et 95% en introduisant des macro-opérateurs d'ordre 2 en fonction des domaines et être compris entre 77% et 98% avec l'ordre idéal calculé. De plus, les gains augmentent encore lorsque la difficulté des problèmes augmente, quel que soit le domaine. En ce qui concerne la qualité des plans, on constate une légère perte de qualité d'environ 10% en moyenne sur les différents domaines testés.

8.4.3 Influence de l'ordre des n-grammes

Nous proposons dans ce paragraphe de regarder plus en détail le gain moyen en temps que l'on peut espérer en fonction de l'ordre des macro-opérateurs introduits. L'objectif est ici de déterminer si notre estimation de l'ordre idéal est bonne et dans quelle mesure l'ordre des macro-opérateurs à ajouter est dépendant du domaine. Les résultats sont donnés à la figure 8.3.

Domaines	Ordre des macros	T $\pm$ t	Q $\pm$ q
Blocksworld	2	66 $\pm$ 21	-7 $\pm$ 0
Blocksworld	3	77 $\pm$ 26	-11 $\pm$ 1
Ferry	2	95 $\pm$ 0	-8 $\pm$ 0
Ferry	4	98 $\pm$ 0	-22 $\pm$ 3
Satellite	2	65 $\pm$ 24	-2 $\pm$ 0
Satellite	3	82 $\pm$ 5	-7 $\pm$ 0
Gripper*	2	68 $\pm$12	-7 $\pm$ 0
Gripper*	3	96 $\pm$ 0	-9 $\pm$ 1
Grid	2	93 $\pm$ 1	-4 $\pm$ 0
Grid	5	97 $\pm$ 0	-42 $\pm$ 5
Depot	2	83 $\pm$ 4	-4 $\pm$ 1
Depot	4	92 $\pm$ 1	-16 $\pm$ 1
Mprime	2	60 $\pm$ 34	-3 $\pm$ 0
Mprime	3	86 $\pm$ 5	-4 $\pm$ 0
Parking*	2	74 $\pm$ 10	2 $\pm$ 2
Parking*	3	79 $\pm$ 6	0 $\pm$ 1
Barman	2	14 $\pm$ 3	-2 $\pm$ 0
Barman	6	80 $\pm$ 11	-9 $\pm$ 1

* Domaines avec macro-opérateurs spécialisés

TABLE 8.2 – Gain par rapport à l'approche sans macros exprimé en pourcentage : T représente le gain en temps et Q le gain en longueur de plan ; t et q représente la dispersion

Les résultats montrent que notre estimation de l'ordre idéal est pertinente. Il est également intéressant de noter (1) que certains domaines, en particulier Ferry et Grid, sont moins sensibles à l'ordre des macro-opérateurs choisis et (2) que le gain en performance décroît rapidement pour la plupart des domaines lorsque l'ordre des macro-opérateurs dépasse 3. Le domaine Barman fait exception. Les performances augmentent avec l'ordre des macro-opérateurs alors que c'est le domaine avec le plus actions instanciées et comportant les opérateurs avec le plus de paramètres.

8.4.4 Discussion sur les résultats

Les points importants que nous pouvons retenir des expérimentations menées sont les suivants :

— La généralisation des séquences d'actions hautement probables en macro-opérateurs améliore de manière significative les performances d'un planificateur pour résoudre de nouveaux problèmes. Toutefois, le mécanisme a tendance à dégrader légèrement la qualité des plans solutions trouvés (cf. tableau 8.2).

— Apprendre sur des problèmes plus simples améliore les performances de la résolution de problèmes plus compliqués. En outre, les gains de performance croissent en fonction de la difficulté des problèmes. Ceci peut s'expliquer par le fait que plus les problèmes sont compliqués plus les macro-actions sont utilisées.

— La technique des n-grammes fournit un cadre formel à la génération de macro-opérateurs ayant les bonnes propriétés sans recourir à une évaluation

hors ligne. Nous avons également montré qu'il était possible d'obtenir une bonne estimation de l'ordre idéal pour les macro-opérateurs pour un domaine donné.

8.5 Conclusion

Nous avons proposé dans ce chapitre une approche heuristique pour améliorer les performances du processus de planification indépendante du domaine ainsi que du planificateur utilisé. Cette méthode permet de tirer partie des problèmes précédemment résolus pour générer des connaissances réutilisables pour la résolution de nouveaux problèmes en s'appuyant sur une technique classiquement utilisée en traitement automatique de la langue naturelle, les n-grammes. Les connaissances du planificateur capitalisées sont encodées sous la forme d'une bibliothèque de macro-opérateurs qui, une fois filtrés et ajoutés au domaine de planification originel, améliorent de manière significative ses performances. Nous avons montré que notre approche était capable d'apprendre des macro-opérateurs ayant les bonnes propriétés, réduisant la profondeur de l'espace de recherche tout en minimisant l'augmentation du coefficient de branchement induit par l'ajout de macro-opérateurs au domaine.

Cette approche ouvre la voie à plusieurs extensions. Le fondement de ces extensions consiste à remplacer la technique des n-grammes par des techniques plus performantes et plus générales issues du domaine de la fouille de données (HAN, KAMBER et PEI, 2011). Ce domaine de recherche a en effet récemment développé de nombreux algorithmes efficaces pour détecter des patterns dans des données séquentielles (FOURNIER-VIGER et al., 2017). Les patterns recherchés par ces algorithmes peuvent avoir des propriétés intéressantes pour l'élaboration de macro-opérateurs pour la planification. Ils peuvent notamment détecter des patterns séquentiels qui sont à la fois très fréquents et, mais également qui sont maximaux dans le sens où ils ne peuvent contenir eux-mêmes d'autres patterns, modélisant ainsi la connaissance de manière compacte. Nous avons débuté des travaux dans cette direction avec le co-encadrement d'une thèse sur le sujet (CASTELLANOS-PAEZ et al., 2016). De plus, ces algorithmes permettent de détecter des patterns séquentiels qui ne sont pas de simples séquences d'actions contiguës, comme c'est le cas pour la technique des n-grammes, mais des séquences d'ensembles d'actions potentiellement non ordonnées et non contiguës. Ceci laisse envisager le développement de macro-opérateurs très expressifs pouvant même exprimer une certaine forme de concurrence entre les opérateurs atomiques qui le composent pouvant s'appliquer dans le cadre de la planification distribuée.

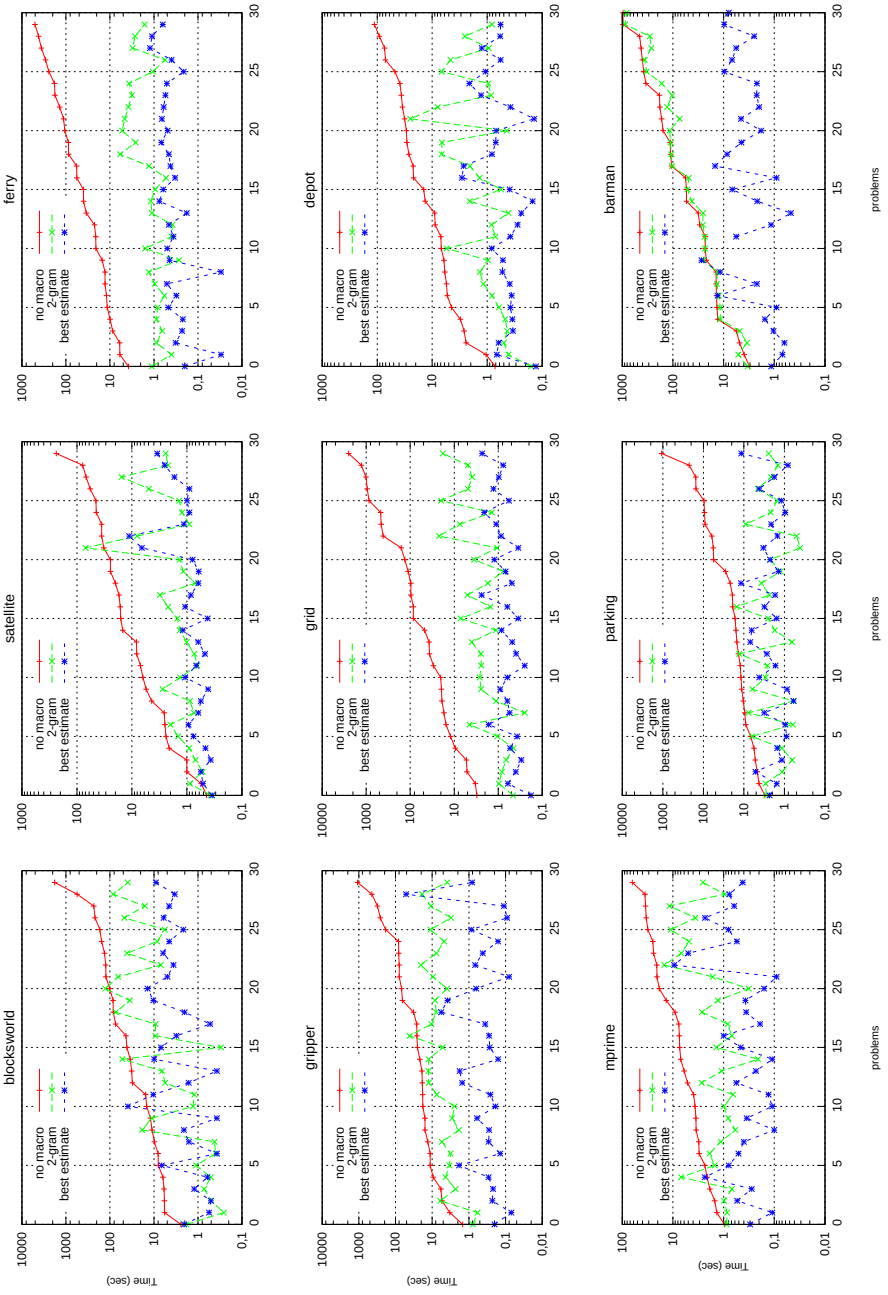


FIGURE 8.2 – Performances en temps obtenues sans macro-opérateurs, avec macro-opérateurs d'ordre 2 et avec macro-opérateurs d'ordre idéal n^* (*best estimate order*). Les problèmes sont classés en fonction de leur difficulté à être résolus sans macro-opérateurs.

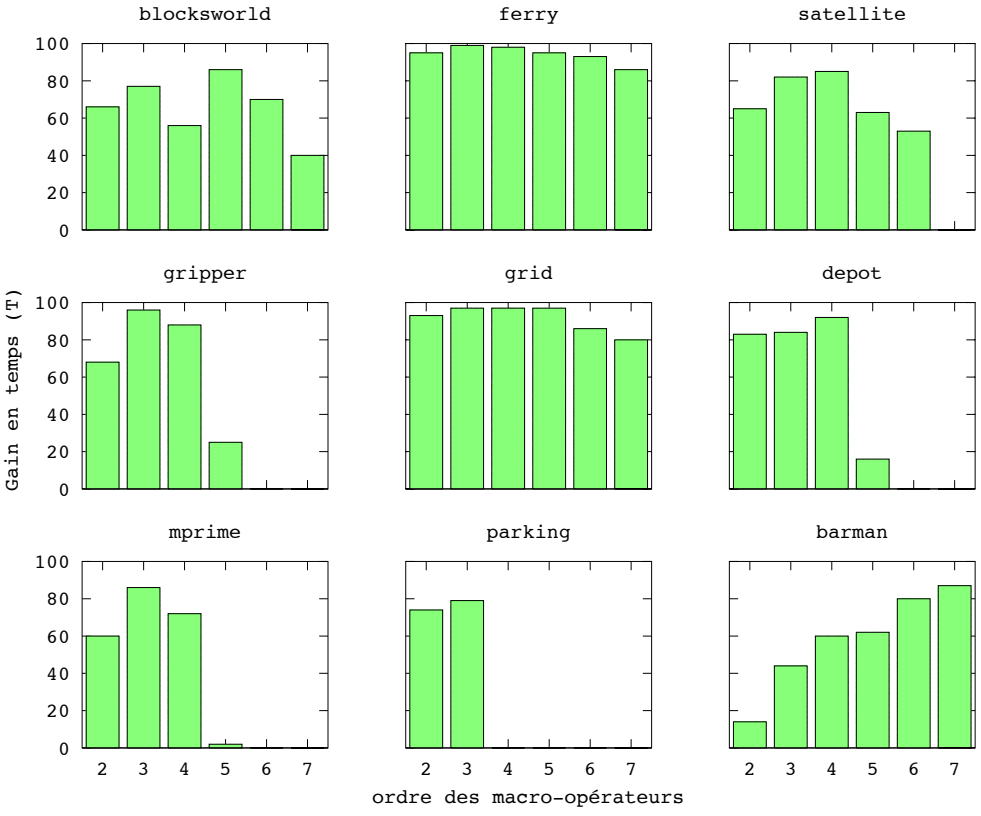


FIGURE 8.3 – Gain de performances en temps (T) en fonction de l'ordre de macro-opérateurs de 2 à 7 exprimé en pourcentage

Chapitre 9

Une méthode heuristique pour l'encodage et la simplification des problèmes HTN

PARMI l'ensemble des techniques de planification, la planification HTN (*Hierarchical Task Network*) est l'une des techniques les plus utilisées en pratique (BEVACQUA et al., 2015 ; KICHKAYLO et al., 2007 ; LALLEMENT, DE SILVA et ALAMI, 2014 ; STRENZKE et SCHULTE, 2011 ; WESER, OFF et ZHANG, 2010) pour des raisons d'efficacité, mais aussi pour l'expressivité des langages HTN. En effet, les langages HTN permettent de spécifier des connaissances métiers de haut niveau d'abstraction, pouvant être utilisées par les algorithmes de recherche pour guider de manière efficace la synthèse d'un plan solution. C'est cette raison notamment qui nous a amenés à utiliser déjà cette technique au chapitre 6.

Contrairement à la planification STRIPS (FIKES et NILSSON, 1971), où le but est défini comme un ensemble de propositions à atteindre, en planification HTN, le but s'exprime comme une tâche ou un ensemble de tâches à réaliser auxquelles il est possible d'associer des contraintes. Ce couple *(tâches, contraintes)* est appelé un *réseau de tâches*. La recherche d'un plan solution consiste à décomposer le réseau de tâches initiales définissant le but, en respectant les contraintes spécifiées, en un ensemble de sous-tâches primitives qui peuvent être exécutées par une action au sens classique de la planification. La décomposition est réalisée en appliquant des règles de décomposition définies par des opérateurs hiérarchiques de planification appelées *méthodes*. Chaque méthode définit une décomposition possible d'une tâche en un ensemble de sous-tâches avec les contraintes qui les lient. La décomposition se termine lorsque le processus de décomposition aboutit à un réseau de tâches contenant uniquement des tâches primitives exécutables par une action et dont l'ensemble des contraintes sont vérifiées.

Parallèlement au développement de la planification HTN, de nombreux algorithmes de planification performants n'utilisant pas de représentation hiérarchique ont été développés, e.g., FastForward (HOFFMANN et NEBEL, 2001) que nous avons déjà cité ou encore FastDownward (HELMERT, 2006). Ils reposent tous sur une étape de prétraitement qui consiste à dénombrer les actions possibles à partir des opérateurs de planification définis dans le domaine. Cette étape est cruciale pour ces algorithmes pour plusieurs raisons. Tout d'abord, cette étape de dénombrement, ou d'instanciation, réduit le nombre d'actions du problème de planification grâce à des mécanismes de simplification. Ceci a pour conséquence de réduire le coefficient de branchement de l'espace de recherche. En second lieu, le fait de dénombrer l'ensemble des actions d'un problème de planification permet de réaliser une étude *a*

priori des propriétés du monde atteignables. Cette étude est un préalable indispensable à l'élaboration et au développement d'heuristiques efficaces pour guider la recherche d'un plan solution (HASLUM, BONET et GEFFNER, 2005 ; HASLUM et GEFFNER, 2000 ; HELMERT et al., 2014 ; HOFFMANN et NEBEL, 2001 ; HOFFMANN, PORTEOUS et SEBASTIA, 2004 ; RICHTER et WESTPHAL, 2008). Troisièmement, cette étape de prétraitement est un prérequis nécessaire à l'encodage d'un problème de planification dans des formalismes tels que CSP (BARTÁK, SALIDO et ROSSI, 2010 ; KAMBHAMPATI, 2000) ou SAT (KAUTZ et SELMAN, 1992 ; RINTANEN, 2012). Pour toutes ces raisons, il serait intéressant d'appliquer le même prétraitement aux problèmes HTN afin de réduire le nombre de décompositions possibles et ainsi la taille de leur espace de recherche.

Dans ce contexte, notre contribution consiste à proposer dans ce chapitre une extension des mécanismes d'instanciation classiques à la planification HTN (RAMOUL et al., 2016, 2017). Ce travail fait l'objet d'une thèse en cours. Dans une première section, nous présentons brièvement l'état de l'art de la planification HTN avant de définir les concepts nécessaires à la présentation de notre contribution. Nous détaillons ensuite les mécanismes d'instanciation et de simplification développés pour la planification HTN et nous terminons par leur évaluation.

9.1 Un état de l'art sur la planification hiérarchique

L'utilisation d'une représentation hiérarchique des actions est une vieille idée en planification. Déjà en 1975, (SACERDOTI, 1975) proposait un planificateur appelé NOAH (*Nets of Action Hierarchies*) fondé sur ce principe. Son planificateur était construit sur une structure de données appelée *procedure net*. Cette structure de données introduisait pour la première fois le concept de réseau de tâches et de décomposition que nous retrouvons de nos jours dans la plupart des planificateurs HTN modernes. Chaque réseau de tâches représente une hiérarchie de nœuds où chaque nœud peut être soit une action primitive soit une méthode traduisant la décomposition d'une tâche en sous-tâches. Les nœuds à chaque niveau de la hiérarchie sont reliés par des contraintes de précédence. L'algorithme de recherche implanté dans NOAH fonctionne sur le principe de la décomposition d'un réseau de tâches initial en réseau de tâches pouvant être exécutées par des actions primitives tout en respectant les contraintes de précédence entre tâches. De nombreux planificateurs se sont par la suite appuyés sur ces travaux.

On peut citer parmi les premiers planificateurs hiérarchiques : (1) Noline, développé par (TATE, 1977), qui étend l'approche de Sacerdoti et introduit un formalisme pour décrire des domaines hiérarchiques ; (2) O-Plan (CURRIE et TATE, 1991) et son successeur O-Plan2 (TATE, DRABBLE et KIRBY, 1994), qui proposent pour la première fois des heuristiques pour guider la décomposition hiérarchique en intégrant les concepts de typage, de temps ainsi que la gestion de ressources ; (3) SIPE (*System for Interactive Planning and Execution*) (WILKINS, 1984) et son successeur SIPE2 (WILKINS, 1990), qui sont des planificateurs hiérarchiques complets qui apportent de nombreuses améliorations à NOAH. Ils permettent notamment d'entrelacer planification et exécution, de développer des plans de manière interactive, de replanifier en cas d'échec au cours de l'exécution du plan et d'utiliser des actions avec des effets conditionnels ; (4) UMCP (*Universal Method Composition Planner*) (EROL, HENDLER et NAU, 1994), qui est le premier planificateur HTN dont l'algorithme de recherche a été montré comme étant correct et complet.

La caractéristique commune de tous ces planificateurs est la nature de l'espace de recherche qu'ils explorent pour trouver un plan solution. En effet, ils explorent tous un espace de plans. Aucun état n'est maintenu en mémoire pendant la recherche. À chaque décomposition, le réseau de tâches résultant est vu comme un plan partiel. Le principal avantage de cette approche est qu'elle génère des plans d'actions partiellement ordonnées beaucoup plus souples que les plans d'actions totalement ordonnés, laissant le choix définitif de l'ordre des actions à exécuter au moment de l'exécution du plan.

Par opposition, les planificateurs HTN modernes tels que SHOP (*Simple Hierarchical Ordered Planner*) (NAU et al., 1999) maintiennent un état pendant la recherche d'un plan solution et explorent un espace d'états. Chaque réseau de tâches contient, en plus des contraintes de précédence entre tâches, un état qui représente l'état du monde. Une décomposition est applicable à un réseau de tâches si et seulement si les préconditions de la décomposition sont vérifiées dans l'état défini par le réseau de tâches. Cette approche est très puissante. Elle élague très rapidement des décompositions non prometteuses et a donné lieu à de nombreux travaux. Par exemple, SHOP a été étendu pour générer des plans partiellement ordonnés (NAU et al., 2003), des plans temporels (ASUNCIÓN et al., 2005) ou encore capables d'intégrer des préférences (SOHRABI, BAIER et MCILRAITH, 2009). SHOP a également été étendu dans un contexte multi-agent (PELLIER et FIORINO, 2007) ou encore pour résoudre des problèmes non déterministes (KUTER et al., 2009). Finalement, le planificateur GoDel (SHIVASHANKAR et al., 2013) a essayé de faire le lien entre planification classique et planification hiérarchique. Ce planificateur est capable d'exploiter les informations hiérarchiques lorsqu'elles sont disponibles pour accélérer la recherche d'un plan solution et de fonctionner en chaînage avant en l'absence d'information.

Pour une analyse détaillée des différentes techniques de planification HTN, nous invitons le lecteur à se reporter à (GEORGIEVSKI et MARCO, 2015). Pour une analyse de complexité, nous recommandons la lecture des articles suivants : (EROL, HENDLER et NAU, 1994) et (ALFORD, BERCHER et AHA, 2015).

9.2 Planification HTN définitions et concepts

9.2.1 Opérateurs, méthodes et tâches

Contrairement à la planification classique, la planification HTN repose sur deux types d'opérateurs de planification : des opérateurs classiques au sens STRIPS déjà définis et des opérateurs qui expriment la décomposition du problème en sous-tâches appelées des *méthodes*.

Définition 9.1 (Méthode)**.** *Une* méthode *est un triplet* $m = (name(m), pre(m), subtasks(m))$ *où*

- *name*(m) *est une expression syntaxique de la forme* $t(u_1, \ldots, u_k)$ *où* t *est le nom de la méthode et* $u_1, \ldots, u_2$ *ses paramètres ;*
- $pre(m)$ *une expression logique qui définit les préconditions de la méthode, i.e., les conditions qui doivent être vérifiées pour appliquer la méthode ;*
- $subtask(m)$ *la liste des sous-tâches à effectuer pour appliquer la méthode.*

Définition 9.2 (Décomposition)**.** *Une* décomposition *est une méthode complètement instanciée. Une décomposition* d *est applicable dans un état* s *si* $pre^+(d) \in s$ *et* $pre^-(d) \cap s = \emptyset$.

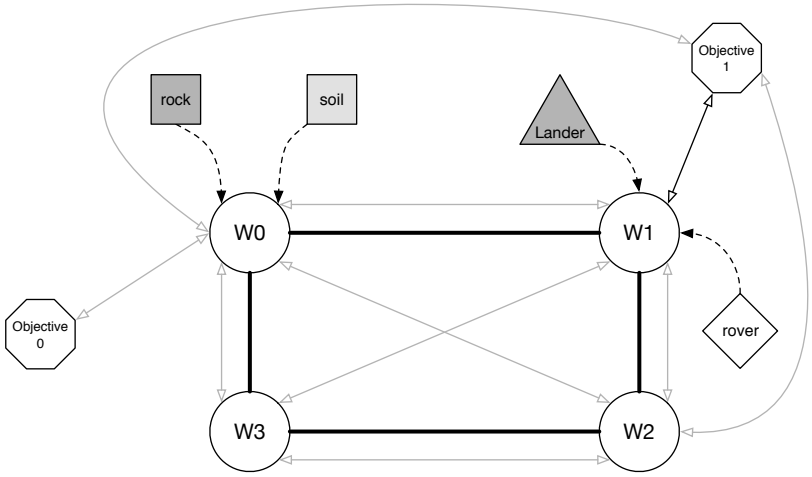


FIGURE 9.1 – Exemple de problème du monde des rovers martiens

Définition 9.3 (Tâche)**.** *Une* tâche *est une expression syntaxique de la forme* $t(u_1, \ldots, u_k)$ *où* t *est le nom de la tâche et* $u_1, \ldots, u_k$ *les paramètres de la tâche. Une tâche est dite complètement instanciée si tous ses paramètres sont instanciés.*

Une action a accomplit une tâche $t(u_1, \ldots, u_k)$ dans un état s si $name(a) = t(u_1, \ldots, u_k)$ et a est applicable dans s. Dans ce cas, nous qualifierons t de tâche *primitive*. De manière analogue, une décomposition d accomplit une tâche $t(u_1, \ldots, u_k)$ dans un état s si $name(d) = t(u_1, \ldots, u_k)$ et d est applicable dans s. Dans ce cas, nous qualifierons t de tâche *non primitive*.

Exemple 9.1 (Le monde des rovers martiens)**.** Pour illustrer ces définitions, considérons le domaine des rovers martiens inspiré du domaine Rovers des compétitions internationales de compétition IPC exprimé de manière hiérarchique. Des rovers doivent se rendre dans différents endroits appelés *waypoints* pour collecter des échantillons de roche et de sol, prendre des photos, et le cas échéant, transmettre les données à leur base de rattachement.

La figure 9.1 donne un exemple d'un problème simple comportant quatre *waypoints* w_0, w_1, w_2, w_3 connectés : le seul rover du problème peut se déplacer de w_0 à w_1, de w_1 à w_2, etc. Le rover est positionné initialement au *waypoint* w_1. Dans ce problème, w_1 est considéré également comme la base de rattachement. Les échantillons de roche et de sol à collecter sont situés en w_0. Les doubles flèches sur la figure représentent les contraintes de visibilité entre les *waypoints* et les objectifs à photographier, par exemple : *(visible* w_0 w_2*)*, *(visible_from* $objective_0$ w_0*)*, etc. Le but du problème est de collecter des échantillons de roche et de sol se trouvant en w_0 et de transmettre à la base une image de l'objectif $objective_1$. Ce but peut s'exprimer par la liste de tâches suivantes : *(get_soil_data* w_0*)*, *(get_rock_data* w_0*)*, *(get_image_data* $objective_1$ *low_res)*.

L'opérateur *navigate* du domaine Rovers de notre exemple est donné ci-dessous dans le langage PDDL :

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

L'opérateur *navigate* possède trois paramètres : *?x* de type rover, *?from* et *?to* de type *waypoint*. Il définit l'action de déplacer un rover *?x* d'un *waypoint* initial *?from* à un *waypoint* destination *?to*. Les préconditions de l'opérateur *navigate* expriment le fait que le rover *?x* doit être préalablement disponible au *waypoint ?from*, que le *waypoint ?to* doit être visible depuis *?from*, et que *?x* peut se déplacer entre *?from* et *?to*. Les effets de l'opérateur expriment qu'après son exécution *?x* sera au *waypoint ?to* et ne sera plus au *waypoint ?from*.

Considérons maintenant une méthode *do_navigate* de notre domaine Rovers exprimée dans un langage proche de PDDL permettant de généraliser le déplacement d'un rover *?x* d'un *waypoint* initial *?from* à un *waypoint ?to* qui ne sont pas visibles contrairement à l'opérateur *navigate* :

```
(:method do_navigate
 :parameters (?x - rover ?from ?to - waypoint)
 :precondition (and (not (can_traverse ?x ?from ?to))
                    (not (visited ?mid))
                    (can_traverse ?x ?from ?mid))
 :subtasks ((navigate ?x ?from ?mid) (visit ?mid)
            (do_navigate ?x ?mid ?to) (unvisited ?mid)))
```

La méthode *do_navigate* possède les mêmes paramètres que *navigate*. Cette méthode est applicable si (1) le rover *?x* ne peut pas se déplacer directement de *?from* à *?to*, (2) le rover n'a pas visité un *waypoint* intermédiaire *?mid* et finalement (3) le rover doit être capable de se déplacer vers ce *waypoint* intermédiaire. L'exécution de cette méthode requiert l'exécution récursive de la séquence de tâches suivantes : *(navigate ?x ?from ?mid)*, *(visit ?mid)*, *(do_navigate ?x ?mid ?to)* and *(unvisited ?mid)*. Notons que contrairement aux opérateurs, les méthodes peuvent utiliser des paramètres non déclarés (dans notre exemple *?mid*). Le type de ces paramètres non déclarés doit être inféré préalablement à leur instanciation.

9.2.2 Problèmes et solutions

Définition 9.4 (Problème de planification HTN)**.** *Un* problème de planification HTN *est un tuple* $P = (s_0, T, O, M)$ *où*

- s_0 *est l'état initial définissant l'état du monde connu ;*
- T *est la liste des tâches initiales qui doivent être exécutées et qui représentent le but à atteindre ;*
- O *est l'ensemble des opérateurs du problème ;*
- M *est l'ensemble des méthodes du problème définissant les décompositions des tâches en sous-tâches.*

Définissons maintenant ce que signifie pour un plan $\pi = \langle a_1, \ldots, a_2 \rangle$ d'être solution d'un problème de planification $P = (s_0, T, O, M)$. En d'autres termes que signifie d'accomplir la liste des tâches T de P. Intuitivement, cela implique de trouver une décomposition de T en un ensemble de sous-tâches primitives ordonnées chacune pouvant être accomplie par une action du problème et applicables dans l'état initial s_0. La définition d'un plan solution peut s'exprimer de manière récursive en considérant trois cas.

Définition 9.5 (Plan solution)**.** *Un plan* $\pi = \langle a_1, \ldots, a_n \rangle$ *est solution d'un problème de planification* $P = (s_0, T, O, M)$ *ssi :*

Cas 1. T *est une liste de tâches vide. Dans ce cas, le plan vide* $\pi = \langle\rangle$ *est solution du problème* P*.*

Algorithme 16 : HTN(s_0, T, O, M)

```
1  Let T = ⟨t_1, ..., t_k⟩
2  if k = 0 then return the empty plan ⟨⟩
3  if t_1 is primitive then
4      A ← {a | a is an action obtained by grounding an operator
         o ∈ O such that a accomplishes t_1 and a is applicable in s_0}
5      if A = ∅ then return failure
6      non-deterministically choose an action a ∈ A
7      π ← HTN(γ(s_0, a), ⟨t_2, ..., t_k⟩, O, M)
8      if π = failure then return failure
9      else return a ⊕ π
10 else if t_1 is a non-primitive task then
11     D ← {d | d is a decomposition obtained by grounding a method
         m ∈ M such that d accomplishes t_1 and d is applicable in s_0}
12     if D = ∅ then return failure
13     non-deterministically choose a decomposition d ∈ D
14     return HTN(s_0, subtasks(d) ⊕ ⟨t_2, ..., t_k⟩, O, M)
```

Cas 2. *La première tâche de $t \in T$ est primitive. Dans ce cas, π est solution de P s'il existe une action a obtenue par instanciation d'un opérateur $o \in O$ telle que (1) a accomplit t, (2) a est applicable dans s_0 et (3) $\pi = \langle a_2, \ldots, a_n \rangle$ est solution du problème de planification HTN :*

$$P' = (\gamma(s_0, a_1), T - \{t\}, O, M)$$

Cas 3. *La première tâche $t \in T$ n'est pas primitive. Dans ce cas, π est solution s'il existe une décomposition d obtenue par instanciation d'une méthode $m \in M$ telle que (1) d accomplit t et (2) π est solution du problème HTN :*

$$P' = (s_0, T - \{t\} \cup \mathit{subtasks}(d), O, M)$$

9.2.3 Procédure générique de planification hiérarchique

Une procédure générique et non déterministe pour résoudre un problème de planification HTN est décrite par l'algorithme 16. La procédure s'appuie sur la définition récursive d'un plan solution que nous venons de donner.

Cette procédure prend en paramètre d'entrée un problème de planification HTN $P = (s_0, T, O, M)$. La procédure vérifie dans un premier temps (ligne 2) que la liste des tâches du problème T n'est pas vide. Si c'est le cas, aucune tâche ne doit être exécutée et le plan solution vide est retourné. Dans le cas contraire, deux cas doivent être considérés en fonction du type de la première tâche t_1 de T.

Cas 1. Si t_1 est primitive (ligne 3) alors la procédure calcule l'ensemble A des actions qui accomplissent t_1 et qui sont applicables dans l'état initial s_0 (ligne 4). Si cet ensemble est vide (ligne 5), cela signifie qu'aucune action ne permet d'accomplir t_1 dans l'état courant. La procédure retourne donc un échec. Dans le cas contraire, la procédure choisit de manière non déterministe une action $a \in A$ (ligne 6) et s'appelle récursivement pour trouver un plan solution π au problème $P' = (\gamma(s_0, a), T - \{t_1\}, O, M)$ (ligne 7). Finalement, si P'

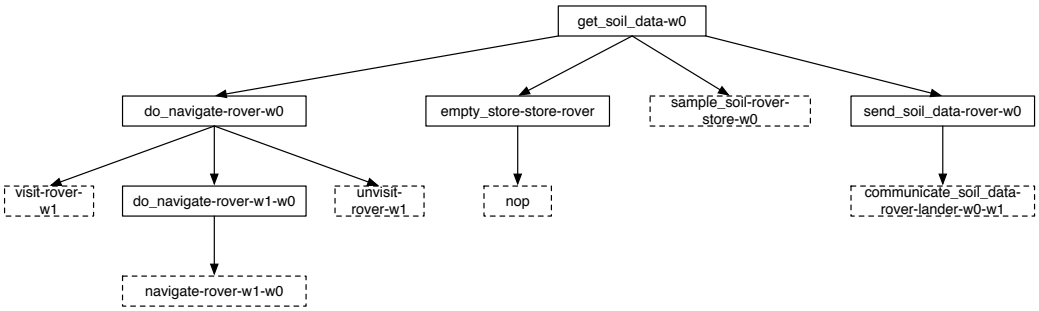


FIGURE 9.2 – Exemple de décomposition totalement ordonnée d'une tâche non-primitive en sous-tâches primitives : les tâches primitives (respectivement non-primitives) sont représentées par des boîtes avec des contours aux traits pointillés (respectivement pleins).

n'a pas de solution la procédure retourne échec (ligne 8) sinon elle retourne comme plan solution la concaténation de l'action a avec le plan solution π.

Cas 2. Si t_1 est non-primitive (ligne 10) alors la procédure calcule l'ensemble des décompositions possibles pour accomplir t_1 qui sont applicables dans s_0 (ligne 11). Si aucune décomposition n'existe pour accomplir la tâche t_1 (ligne 12) alors la procédure retourne échec. Dans le cas contraire, la procédure choisit de manière non déterministe une décomposition d pour accomplir t_1 (ligne 13) et retourne récursivement le plan solution du problème $P' = (s_0, \mathit{subtasks}(d) \oplus \langle t_2, \ldots, t_k \rangle, O, M)$ (ligne 14).

Un exemple d'arbre de décompositions obtenu par la procédure HTN est représenté à la figure 9.2. Cet arbre représente la décomposition de la tâche *get_soil_data-w*$_0$ en tâches primitives. Au premier niveau de décomposition, la tâche *get_soil_data-w*$_0$ est décomposée en 3 sous-tâches non primitives : (1) *do_navigate-rover-w*$_0$, (2) *empty_store-store-rover* et (3) *send_soil_data-rover-w0* et une tâche primitive *sample_soil-rover-store-w*$_0$. Un plan solution possible pour la tâche *get_soil_data-w*$_0$ est : $\langle$*(visit-rover-w*$_1$*)*, *(navigate-rover-w*$_1$*-w*$_0$*)*, *(unvisit-rover-w*$_1$*)*, *(sample_soil-rover-store-w*$_0$*)*, *(communicate_soil_data-rover-lander-w*$_0$*-w*$_1$*)*$\rangle$.

9.3 L'instanciation des problèmes de planification HTN

Le processus d'instanciation et de simplification que nous proposons pour la planification HTN est présenté à la figure 9.3. Il prend en entrée un domaine et un problème HTN et produit un problème HTN simplifié et complètement instancié. Ce processus comprend trois étapes : une étape qui consiste à calculer ce que nous appelons des *inerties* (nous expliciterons le concept d'inertie au paragraphe 9.3.2) ; une étape d'instanciation des opérateurs commune à la plupart des planificateurs non HTN et une étape d'instanciation des méthodes.

9.3.1 Des éléments de complexité

L'instanciation consiste à dénombrer l'ensemble des actions et des décompositions possibles à partir de la définition des opérateurs et des méthodes d'un problème de planification HTN ainsi que des objets déclarés dans le problème. Le nombre d'instances dépend du nombre de paramètres éventuellement typés de l'opérateur ou de la méthode et du nombre d'objets du problème. Soit x_i le

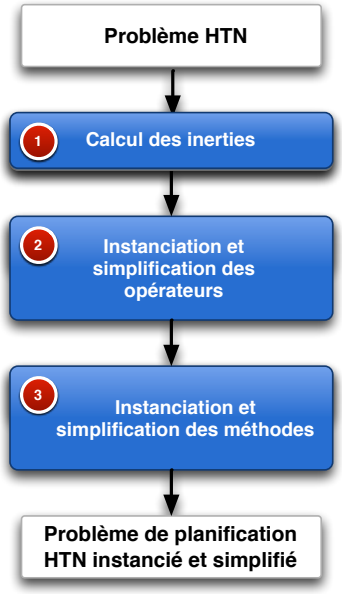


FIGURE 9.3 – Vue globale des différentes étapes du processus d'instanciation et de simplification des problèmes de planification HTN

paramètre d'un opérateur et $D(x_i)$ son domaine, i.e., l'ensemble des objets ayant le type de x_i. Le nombre d'instances d'un opérateur o possédant n paramètres $x_1, \ldots, x_n$ est égale à $\prod_{i=1}^{n} |D(x_i)|$. Ce nombre augmente donc très rapidement avec le nombre de paramètres. Par exemple, le nombre d'actions résultant de l'instanciation du seul opérateur *(communicate_soil_data ?x - rover ?l - lander ?p1 - waypoint ?p2 - waypoint ?p3 - waypoint)* du domaine Rovers pour le problème 40 de la compétition IPC-5 comportant 14 rovers et 100 *waypoints* produira 14 millions d'instances différentes. Dans le contexte de la planification HTN, il est également nécessaire d'instancier les méthodes dont le nombre de paramètres est généralement plus grand que celui des opérateurs, ce qui a pour conséquence d'augmenter encore le nombre d'instances possibles. À titre d'illustration, la figure 9.4 montre l'évolution du nombre de décompositions obtenues après instanciation sur les 23 premiers problèmes de la compétition IPC-5 du domaine Rovers. Il est donc essentiel de développer des mécanismes de simplifications pour réduire le nombre de méthodes au plus tôt pendant le processus d'instanciation, afin de limiter le coefficient de branchement de l'arbre de recherche produit par la procédure de planification HTN.

9.3.2 Le calcul des inerties

La première étape du processus d'instanciation et de simplification consiste à calculer les inerties du problème de planification. Le concept d'inertie a été défini pour la première fois par (KOEHLER et HOFFMANN, 1999). Une inertie est une proposition définie dans les préconditions ou les effets d'une action. Il existe deux types d'inertie :

1. *Les inerties positives* qui se définissent comme les propositions qui ne sont produites par aucune action du problème. Si une inertie positive n'est pas présente dans l'état initial du problème, elle ne pourra jamais être vérifiée.
2. *Les inerties négatives* qui se définissent comme les propositions qui ne sont supprimées par aucune action du problème. Si une inertie négative est présente dans l'état initial, alors la proposition sera toujours vérifiée.

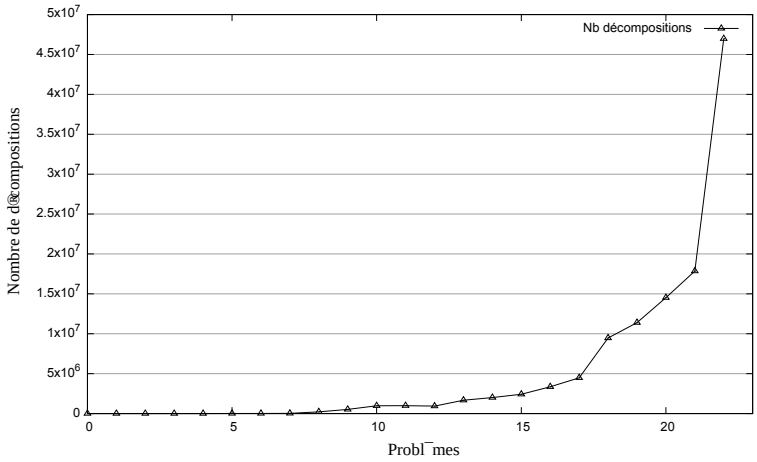


FIGURE 9.4 – Nombre de décompositions obtenues après instanciation des méthodes sur le domaine Rovers IPC-5

Les propositions qui sont à la fois des inerties positives et négatives sont appelées simplement des inerties. Autrement dit, les inerties sont les propositions qui n'évoluent pas. Par opposition, les propositions qui peuvent être produites et consommées sont appelées des *fluents*. Pour plus de détail sur le calcul des inerties, nous invitons le lecteur à se référer à (KOEHLER et HOFFMANN, 1999). Le tableau 9.1 donne les inerties du domaine Rovers. Par exemple, les propositions *can_traverse*, *visible* et *at_lander* n'apparaissent pas dans les effets d'un des opérateurs du domaine Rovers. Ce sont donc des inerties. Les propositions *at_soil_sample* et *at_rock_sample* ne sont jamais produites par un opérateur. Ce sont donc des inerties positives. Les propositions *available*, *communicated_soil_data* et *communicated_rock_data* ne sont jamais supprimées par un opérateur. Ce sont donc des inerties négatives. Finalement, les propositions *at* et *visited* sont consommées et produites. Elles doivent donc être considérées comme des fluents.

9.3.3 Instanciation et simplification des opérateurs

L'instanciation et la simplification des opérateurs pertinents pour le problème de planification se découpent en quatre étapes : une étape de normalisation des expressions logiques contenues dans les opérateurs, une étape d'instanciation proprement dite, une étape de simplification des préconditions et des effets des opérateurs, et enfin, une étape de simplification et de réduction du nombre d'opérateurs du problème de planification.

Étape 1. (Normalisation des opérateurs)

Dans cette étape, toutes les expressions logiques contenant des implications et des quantificateurs sont reformulées sous une forme conjonctive ou disjonctive simple en suivant les règles de réécriture suivantes :

— $\phi \rightarrow \varphi \Rightarrow \neg\phi \wedge \varphi$
— $\neg(\phi \wedge \varphi) \Rightarrow \neg\phi \vee \neg\varphi$
— $forall(?x - type) \Rightarrow x_1 \wedge x_2 \wedge ... \wedge x_n$
— $exists(?x - type) \Rightarrow x_1 \vee x_2 \vee ... \vee x_n$

Proposition	Inertie Positive	Inertie Négative	Type
available	Non	Oui	Inertie négative
at	Non	Non	Fluent
visible	Oui	Oui	Inertie
can_traverse	Oui	Oui	Inertie
communicated_soil_data	Non	Oui	Inertie négative
communicated_rock_data	Non	Oui	Inertie négative
at_soil_sample	Oui	Non	Inertie positive
at_rock_sample	Oui	Non	Inertie positive
at_lander	Oui	Oui	Inertie
visited	Non	Non	Fluent

TABLE 9.1 – Inerties du domaine Rovers

Toutes les formules logiques contenues dans les opérateurs sont affectées par cette réécriture. Ces règles permettent de mettre les expressions logiques sous la forme conjonctive normale, préalable classique à toute manipulation des expressions logiques.

Étape 2. (Instanciation des opérateurs)

Instancier un opérateur consiste à remplacer chaque occurrence d'une variable utilisée dans les expressions d'un opérateur par les objets du même type ou sous-type déclaré dans le problème. Pour chaque combinaison d'objets pouvant être affectée une nouvelle action est créée. Considérons en guise d'illustration, l'action

```
(:action navigate
  :parameters (rover1 waypoint3 waypoint2)
  :precondition
     (and (available rover1) (at rover1 waypoint3)
          (can_traverse rover1 waypoint3 waypoint2)
          (visible waypoint3 waypoint2))
  :effect (and (not(at rover1 waypoint3)) (at rover1 waypoint2))
)
```

est une instance de l'opérateur *navigate* de notre exemple rover où la variable $?x$ a été remplacée par la constante $rover_1$, $?p_1$ par $waypoint_3$ et $?p_2$ par $waypoint_2$. L'instanciation s'appuie sur les types des variables pour déterminer leur domaine de valeurs. Lorsque le domaine n'est pas typé, le type des paramètres de chaque opérateur est inféré préalablement à l'étape d'instanciation à partir des prédicats unaires déclarés dans le domaine.

Étape 3. (Simplification des préconditions et des effets des opérateurs)

La simplification consiste à essayer d'évaluer les propositions contenues dans les opérateurs à *vrai* ou à *faux* en utilisant les inerties du problème. Soit une proposition p contenue dans les préconditions ou les effets d'une action et s_0 l'état initial du problème de planification, la simplification de p suit les règles suivantes :

— **Si** p est une inertie positive et $p \notin I$ **alors** p est simplifiée à **faux**.
— **Si** p est une inertie négative et $p \in I$ **alors** p est simplifiée à **vrai**.
— **Sinon** p ne peut pas être simplifiée.

Toutes les propositions simplifiées peuvent être supprimées du problème de planification. Considérons la proposition *(can_traverse ?x - rover ?p_1 - waypoint ?p_2 - waypoint)* qui n'apparaît ni dans les effets négatifs ni dans les effets positifs d'un opérateur du domaine. Cette proposition est une inertie. Par conséquent, elle peut être remplacée par *vrai* si elle est dans l'état initial du problème ou par *faux* dans le cas contraire. Ceci a pour conséquence de simplifier les préconditions de notre action *navigate* données précédemment. Dans le cas d'une simplification à *faux*, toute l'expression définissant les préconditions de l'action *navigate* : *(and (available rover) (can_traverse rover w1 w0) (visible w1 w0) (at rover w1))* peut être simplifiée à *faux*.

Notons pour terminer que sous certaines conditions non détaillées ici, les formules atomiques contenues dans les préconditions et les effets des opérateurs peuvent être simplifiées même si celles si ne sont pas complètement instanciées réduisant ainsi le coût de l'instanciation.

Étape 4. (Simplification des actions du problème)

La simplification des actions vise à supprimer du problème les actions qui ne pourront jamais être appliquées. Cette étape consiste à simplifier les préconditions et les effets des opérateurs en utilisant les règles classiques de réécriture suivantes :

$$\begin{array}{rclrcl} \neg VRAI & \equiv & FAUX & \neg FAUX & \equiv & VRAI \\ VRAI \wedge \varphi & \equiv & \varphi & \varphi \wedge \varphi & \equiv & \phi \\ FAUX \wedge \varphi & \equiv & FAUX & \varphi \vee \varphi & \equiv & \varphi \\ VRAI \vee \varphi & \equiv & VRAI & \varphi \wedge \neg\varphi & \equiv & FAUX \\ FAUX \vee \varphi & \equiv & \varphi & \varphi \vee \neg\varphi & \equiv & VRAI \end{array}$$

Si le processus de simplifications réduit à *vrai* ou *faux* toute l'expression logique des préconditions ou des effets d'une action, elle peut être simplifiée comme suit :

— Si la précondition ou l'effet d'une action est remplacé par *faux*, l'action est supprimée du problème de planification. Dans le cas où la précondition est fausse, l'action ne peut jamais être appliquée.

— Si tous les effets d'une action sont évalués à *vrai* ou à *faux*, l'action est supprimée puisque l'action ne produit aucun changement.

9.3.4 L'instanciation des méthodes

L'instanciation des méthodes est réalisée en cinq étapes : (1) une étape de normalisation des expressions représentant les préconditions des méthodes, (2) une étape d'inférence des types des paramètres implicites des méthodes, (3) une étape d'instanciation des méthodes, (4) une étape de simplification des préconditions des méthodes, et enfin (5) une étape de simplification et de réduction des méthodes instanciées.

Étape 1. (Normalisation des méthodes)

Cette étape consiste à réécrire les préconditions des méthodes sont sous la forme conjonctive normale. Cette étape est identique à celle réalisée sur les opérateurs.

Étape 2. (Inférence des types des paramètres implicites des méthodes)

La différence avec l'instanciation des opérateurs réside dans le fait que les préconditions des méthodes peuvent spécifier des variables non déclarées dans

les paramètres de la méthode. Dans notre exemple, la variable *?mid* de la méthode *do_navigate* est utilisée dans les sous-tâches et les préconditions de la méthode, mais n'est pas déclarée dans les paramètres de la méthode. Il est donc nécessaire d'inférer son type avant de débuter le processus d'instanciation. Ce processus d'inférence est réalisé en deux étapes :

Étape 2.1 (Inférence des types à partir des tâches)

1. Récupérer toutes les tâches T qui contiennent dans leurs paramètres la variable non déclarée ;
2. Pour chaque tâche $t \in T$, récupérer les opérateurs o ou les méthodes m qui sont pertinents pour t, i.e., tels que $name(o) = t$ ou $name(m) = t$;
3. Pour chaque tâche $t \in T$, récupérer les types déclarés dans les paramètres de o ou de m. Si l'on obtient deux types A et B, où B est un sous-type de A, on infère le type B ;
4. Si plusieurs types n'ayant aucun lien d'héritage sont récupérés, alors une erreur de typage doit être signalée.

Étape 2.2 (Inférence des types à partir des préconditions des méthodes)

1. Récupérer l'ensemble des propositions P, utilisées dans les préconditions de la méthode qui contiennent la variable non déclarée ;
2. Pour chaque proposition $p \in P$, récupérer les types de la variable non déclarée à partir de la déclaration des prédicats du problème de planification ;
3. Si les types obtenus n'ont aucun lien d'héritage, alors une erreur de typage doit être signalée. Sinon, le sous-type feuille de la hiérarchie de types est inféré.

Étape 3. (Instanciation des méthodes)

L'instanciation des méthodes comme celle des opérateurs consiste à remplacer les variables contenues dans la description des méthodes par les objets du problème de même type. Pour chaque combinaison d'objets possibles, une décomposition est produite. Par exemple, une décomposition obtenue à partir de la méthode *do_navigate* est :

```
(:method do_navigate
 :parameters (rover w1 w0)
 :precondition (and (not (can_traverse rover w1 w0))
                    (not (visited w3))
                    (can_traverse rover w1 w3))
 :subtasks ((navigate rover w1 w3) (visit w3)
            (do_navigate rover w3 w0) (unvisit w3)))
```

L'affectation des variables dans cet exemple est le suivant : *?x* est affectée à *rover*, *?from* à w_1, *?to* à w_0, le type inféré de *?mid* est *waypoint* et *?mid* est affectée à w_3.

Étape 4. (Simplification des préconditions des méthodes)

La simplification des préconditions des méthodes est identique à celle réalisée sur les opérateurs.

Étape 5. (Simplification des décompositions du problème

Cette étape consiste à identifier et à supprimer les décompositions obtenues par instanciation des méthodes dont les préconditions ne seront jamais vérifiées quel que soit l'état initial du problème. Deux simplifications sont réalisées :

Étape 5.1 (Simplification fondée sur les préconditions)
La simplification est fondée sur l'évaluation à *vrai* ou à *faux* des préconditions des décompositions. Les préconditions peuvent être évaluées à *vrai* ou *faux* en se fondant sur les règles logiques présentées précédemment. Dans le cas où les préconditions d'une décomposition seraient évaluées à *faux*, la décomposition est supprimée du problème puisque ses préconditions ne seront jamais vérifiées.

Étape 5.2 (Simplification fondée sur les tâches)
La simplification fondée sur les tâches cherche à supprimer les méthodes dont les tâches primitives ne peuvent pas être réalisées. En sachant que la simplification des opérateurs est effectuée avant celle des méthodes, la procédure de simplification est comme suit :

1. Récupérer l'ensemble des tâches T définies dans la décomposition à simplifier ;
2. Pour chaque tâche $t \in T$, vérifier si une action ou une décomposition permet de réaliser t. Si tel n'est pas le cas, alors toute la décomposition est supprimée du problème. Dans le cas contraire, elle est conservée.
3. Itérer le processus tant que le nombre de décompositions du problème diminue.

9.4 Évaluation de l'approche

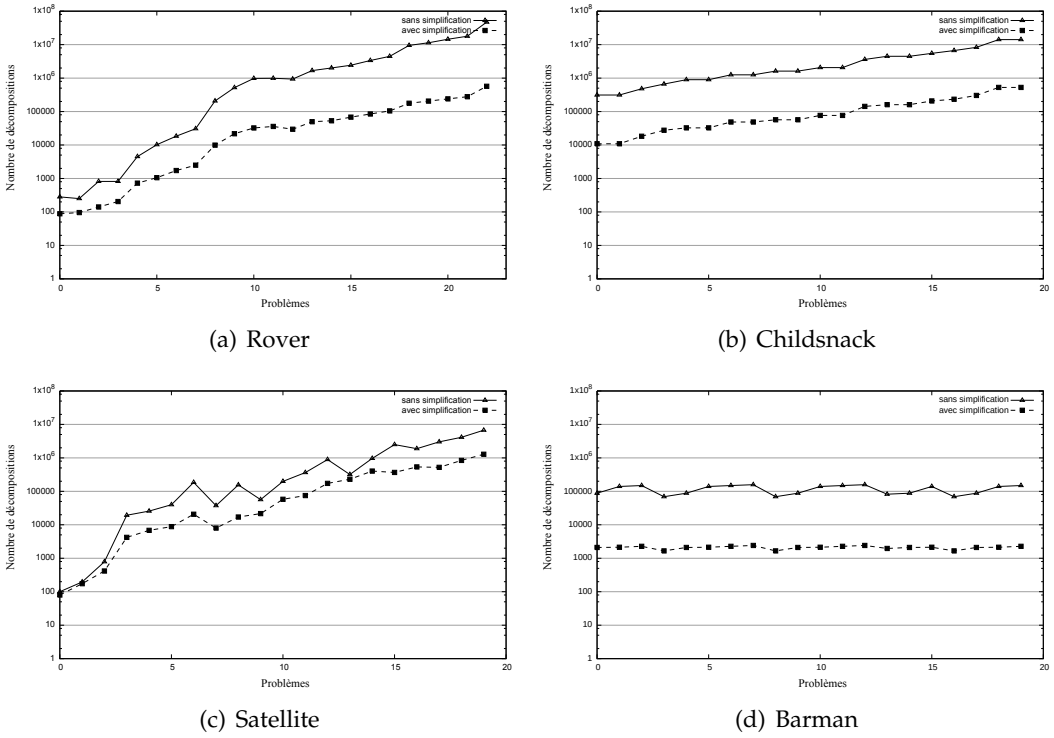
Dans cette partie, nous proposons une évaluation de l'approche proposée selon deux critères :

1. *Un critère statique*, qui consiste à étudier le nombre de décompositions que notre approche a permis d'éliminer par rapport à une instanciation naïve de toutes les méthodes ;
2. *Un critère dynamique*, qui consiste à étudier l'impact sur un algorithme de planification HTN de fonctionner avec un problème de planification HTN complètement instancié et simplifié.

9.4.1 Évaluation du taux de simplification

Le premier critère consiste à regarder le nombre de décompositions obtenues par instanciation des méthodes avec ou sans simplification. Nous avons ici étendu les résultats présentés dans (RAMOUL et al., 2017) sur quatre domaines d'IPC que nous avons exprimés dans le formalisme HTN : Rovers, Childsnack, Satellite et Barman. Chaque jeu de tests contient 20 problèmes classés par ordre croissant de difficulté extraits d'IPC.

La figure 9.5 montre l'évolution du nombre de décompositions obtenues après instanciation avec et sans simplification. Le taux de simplification est défini comme le rapport du nombre de décompositions obtenues sans simplification sur le nombre de décompositions obtenues avec simplification. La figure 9.5 montre que le nombre de décompositions diminue nettement avec notre méthode de simplification, et ce



(a) Rover (b) Childsnack

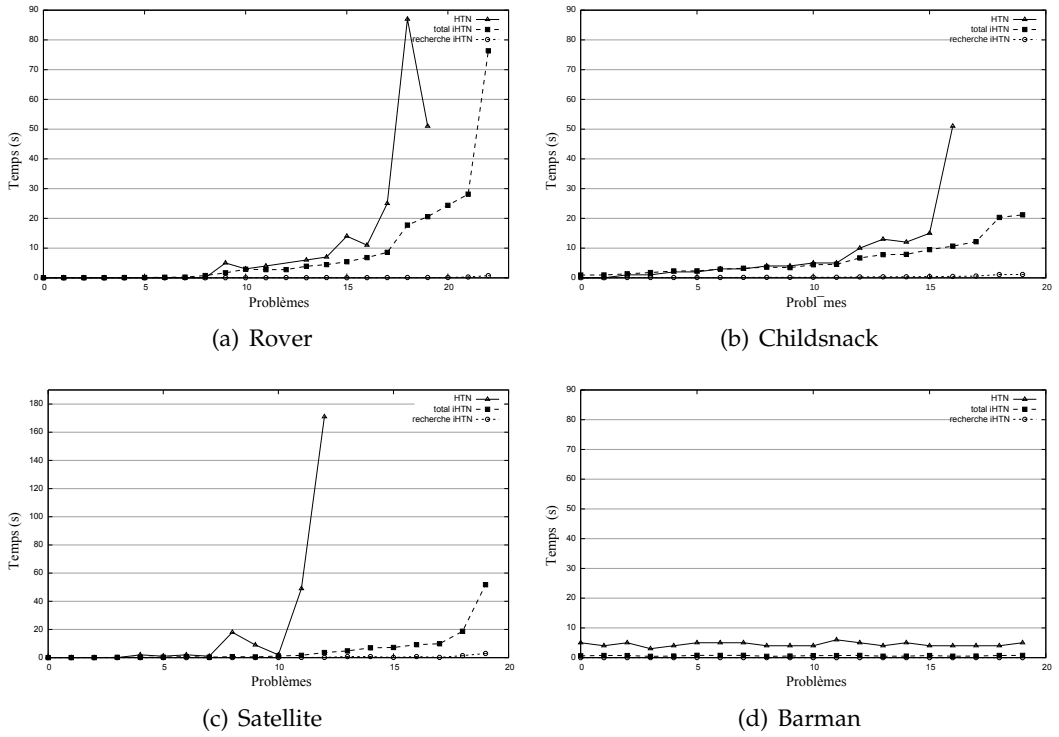
(c) Satellite (d) Barman

FIGURE 9.5 – Représentation logarithmique du nombre de décompositions obtenues après instanciation avec et sans simplification sur les domaines : Rovers, Satellite, Childsnack, Barman extrait d'IPC

d'autant plus que les problèmes sont difficiles. Nous pouvons observer sur les domaines Rovers, Childsnack et Satellite que le nombre de décompositions décroît exponentiellement avec la taille des problèmes. Par exemple, considérons la figure 9.5(a). Le taux de simplification obtenu est de 30,16 pour le problème 10 (997 202 décompositions obtenues sans simplification contre 32 400 avec). Il est de 82,58 pour le problème 22 (46 980,192 décompositions obtenues sans simplification contre 568 850 avec). Pour le domaine barman (cf. figure 9.5(d)), le taux de simplification reste haut et stable compris entre 41,65 et 65,57. Ceci s'explique par le fait que le nombre d'objets des problèmes reste relativement stable malgré la difficulté croissante des problèmes.

9.4.2 Évaluation qualitative et quantitative

Pour confirmer l'impact positif sur les performances de manipuler un problème de planification HTN complètement instancié plutôt qu'un problème qui ne l'est pas, nous avons développé une version de l'algorithme HTN (cf. Algo. 16) capable de fonctionner avec un problème instancié. Nous appellerons cette version de l'algorithme iHTN (*intanciated HTN*) dans la suite de ce chapitre. Les deux versions de l'algorithme HTN et iHTN ont été développées en JAVA en s'appuyant sur la librairie PDDL4J (PELLIER et FIORINO, 2017). Notons que l'algorithme HTN de référence est une implémentation de l'algorithme développé initialement pour le planificateur SHOP (NAU et al., 2003). Finalement, pour avoir un point de comparaison avec un planificateur non HTN, nous avons également réalisé les expérimentations avec le planificateur FastDownard (HELMERT, 2006). Les expérimentations ont été réalisées



(a) Rover (b) Childsnack

(c) Satellite (d) Barman

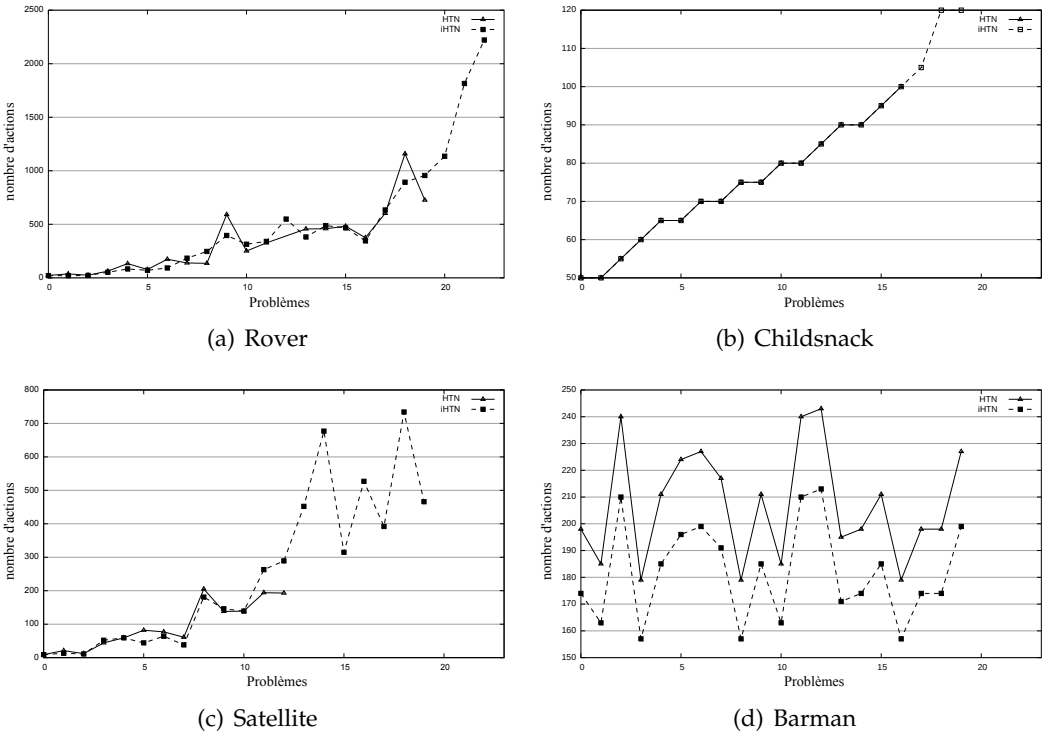
FIGURE 9.6 – Comparaison du temps de recherche entre HTN et iHTN sur les domaines : Rovers, Childsnack, Satellite and Barman

comme précédemment sur les domaines Rovers, Childsnack, Satellite et Barman. Tous les résultats ont été obtenus sur une machine équipée d'un processeur Intel Core I7 cadencé à 2,2GHz avec 16Go de mémoire vive.

Les résultats des expérimentations sont présentés aux figures 9.6 et 9.7 ainsi qu'au tableau 9.2. La figure 9.6 présente les résultats au regard du temps de recherche de HTN et de iHTN. Pour iHTN, nous donnons le temps mis pour effectuer la recherche d'un plan solution et le temps total, prétraitement inclus. La figure 9.7 présente les résultats au regard de la qualité des plans solution trouvés (longueur des plans trouvés). Finalement, le tableau 9.2 propose une comparaison de HTN, iHTN et de FastDownard en termes de temps de recherche en respectant les règles des compétitions IPC (CARLOS, SERGIO et MALTE, 2013) : un score est attribué pour chaque planificateur et pour chaque problème. Ce score est le ratio du temps de cherche mis par le planificateur évalué et du meilleur temps obtenu par les planificateurs comparés. Le meilleur planificateur obtient au plus un score de 1. Un temps limite pour la résolution de chaque problème est fixé à 90 secondes sauf pour le domaine Rovers ou il est fixé à 180 secondes.

Résultats et analyse en termes de temps de recherche

Pour les quatre domaines testés, iHTN est toujours plus performant que la version HTN classique. L'utilisation d'un problème instancié accélère donc bien la recherche d'un plan solution. De plus, la recherche d'un plan solution avec iHTN ne représente que 5% du temps global et n'excède jamais 3 secondes même pour les problèmes très grands. Cela peut être très intéressant si une replanification est nécessaire, notamment si le but change comme étudié au chapitre 4. En effet, dans ce cas, il n'est pas nécessaire de relancer une phase de prétraitement pour résoudre



(a) Rover (b) Childsnack

(c) Satellite (d) Barman

FIGURE 9.7 – Comparaison de la longueur des plans solutions entre HTN et iHTN sur les domaines : Rovers, Childsnack, Satellite and Barman

le problème modifié, car toutes les simplifications réalisées par le prétraitement s'appuient uniquement sur l'état initial du problème.

La figure 9.6 montre également que le gain n'est pas très important pour les petits problèmes, mais croît rapidement avec la taille des problèmes. Finalement, la figure montre aussi que certains problèmes ne sont pas résolus par la version HTN classique alors que la version iHTN résout tous les problèmes de notre jeu de tests.

Résultats et analyse en termes de qualité des plans solutions trouvés

La figure 9.7 montre pour chaque problème la longueur des plans solutions obtenus. Les résultats d'iHTN et d'HTN sont comparables. Les différences marginales observées sont dues au choix aléatoire des décompositions réalisé pendant la recherche. Toutefois, notons que pour le domaine Barman, iHTH est toujours meilleur que HTN.

Résultats et analyse comparative avec à un planificateur en chaînage avant

Le tableau 9.2 présente les résultats de iHTN comparés à un planificateur en chaînage avant de référence FastDownard. iHTN domine FastDownard sur 3 des 4 domaines (Childsnack, Satellite et Barman) et la version classique d'HTN sur tous les domaines. Quels que soient les domaines, il résout tous les problèmes de notre jeu de tests, ce qui n'est pas le cas de FastDownard et de HTN. FastDownard obtient notamment de très mauvais résultats sur le domaine Barman. La version iHTN implémentée en Java obtient donc des résultats comparables avec un planificateur

	Rover			Childsnack			Satellite			Barman		
Pb	FD	iHTN	HTN	FD	iHTN	HTN	FD	iHTN	HTN	FD	iHTN	HTN
00	1	1	1	0	1	1	1	1	1	0	1	0,51
01	1	1	1	0,49	1	1	1	1	1	0	1	0,55
02	1	1	1	0,34	1	0,99	1	1	1	0,38	1	0,52
03	1	1	1	1	0,62	0,61	1	1	1	0	1	0,51
04	1	1	1	1	0,66	0,66	1	1	0,33	0	1	0,51
05	1	1	1	1	0,71	0,73	1	1	0,39	0	1	0,53
06	1	1	1	0	1	0,96	1	1	0,37	0	1	0,54
07	1	1	1	0	1	0,99	1	1	0,38	0	1	0,53
08	1	1	1	0	1	0,91	1	1	0,33	0	1	0,51
09	1	0,92	0,63	1	0,89	0,84	1	1	0,42	0	1	0,51
10	1	0,86	0,79	0	1	0,94	0,84	1	0,75	0	1	0,54
11	1	0,98	0,81	0	1	0,95	1	0,96	0,40	0	1	0,51
12	1	0,72	0,63	0	1	0,83	0,71	1	0,38	0	1	0,53
13	1	0,76	0,64	0	1	0,81	0,53	1	0	0,27	1	0,51
14	1	0,79	0,59	0	1	0,84	0,64	1	0	0	1	0,50
15	1	0,74	0,62	0	1	0,83	0,42	1	0	0	1	0,54
16	1	0,74	0,54	0	1	0,59	0	1	0	0	1	0,52
17	1	0,70	0,47	0	1	0	1	0,66	0	0	1	0,52
18	1	0,70	0,54	0	1	0	0,66	1	0	0	1	0,56
19	1	0,68	0	0	1	0	1	0,60	0	0	1	0,54
20	1	0,70	0	-	-	-	-	-	-	-	-	-
21	1	0,70	0	-	-	-	-	-	-	-	-	-
Total	22	19,00	15,32	4,84	18,89	14,51	16,81	19,22	7,75	0,65	20	10,51

TABLE 9.2 – Scores en temps obtenus par FastDownward, iHTN et HTN selon les critères IPC

écrit dans un langage compilé. Finalement, les gains entre iHTN et HTN vont de 25% sur les domaines Rovers et Childsnack à plus de 60% pour Satellite et Barman.

9.5 Conclusion

Nous avons présenté dans ce chapitre une approche heuristique pour l'instanciation et la simplification des problèmes de planification HTN. Notre contribution repose sur les techniques d'instanciation développées en planification classique. Nous avons démontré que notre contribution améliorait de manière significative les performances des planificateurs HTN et rivalisait avec les planificateurs en chaînage avant. Ces résultats ouvrent de nombreuses pistes de travail :

1. *Le développement d'heuristiques pour la planification HTN.* En effet, le fait de posséder l'ensemble des actions et des décompositions d'un problème HTN offre la possibilité de réaliser une étude d'atteignabilité préalable au développement d'heuristiques.
2. *Le développement de nouveaux encodages SAT et CSP pour la résolution de problèmes HTN.* Peu de travaux se sont intéressés à l'encodage des problèmes HTN sous la forme de problèmes SAT ou CSP. Pourtant les planificateurs s'appuyant sur ce type de solveurs obtiennent de très bons résultats comme le montrent les résultats des dernières compétitions de planification. Tous les

encodages nécessitent une phase d'instanciation qui dorénavant existe pour la planification HTN.

Quatrième partie

Planifier : Outils

Les avancées théoriques réalisées dans le domaine de la planification sont nombreuses. Toutefois, la planification peine à trouver contrairement à d'autres communautés telles que la communauté SAT ou CSP des débouchés industriels. Ceci s'explique en partie par l'absence de planificateur prêt à l'emploi, documenté et intégrable facilement dans une architecture logicielle plus complexe. Même si l'explication technique apporte une première réponse, nous pensons que d'autres facteurs limitent les débouchés industriels de la planification. En particulier, il apparaît assez clairement que l'écriture des domaines de planification est un frein à sa dissémination. Exprimer et transcrire dans un langage logique tel que PDDL des connaissances métiers pour automatiser un processus est une démarche difficile et coûteuse que peu d'entreprises sont prêtes à payer malgré les gains à long terme qui peuvent en découler.

Partant de ces observations, nous proposons dans la dernière partie de ce mémoire de traiter des aspects moins théoriques, mais tout aussi importants de la planification en s'y intéressant sous l'angle des outils. Nous présentons la librairie de planification PDDL4J que nous avons déjà mentionnée à plusieurs reprises dans ce mémoire. Cette librairie a pour objectif de capitaliser l'ensemble des techniques de planification en proposant un outil performant, mais surtout, développé en respectant une démarche logicielle au-delà du simple code de recherche pour faciliter sa dissémination et sa valorisation dans un contexte industriel.

Chapitre 10

PDDL4J : une bibliothèque Open Source pour la planification

PDDL4J (Planning Domain Description Library for Java [1]) est une librairie *open source* plateforme indépendante pour la planification automatique. Nous la développons et la maintenons seul depuis 2006. Écrite en Java, elle a pour objectifs (1) de fournir un ensemble d'outils de planification à l'état de l'art reposant sur le langage PDDL et (2) de faciliter le développement de nouveaux planificateurs et leur intégration dans des outils existants (PELLIER et FIORINO, 2017). PDDL4J a été utilisée avec succès dans plusieurs domaines, e.g., la composition de services Web (BEVILACQUA et al., 2011), la vérification de processus métiers, (SALNITRI, PAJA et GIORGINI, 2015), les jeux (CHENG et al., 2011) ou encore la robotique (ZAMAN et al., 2013). La librairie est actuellement en incubation auprès de la SATT Linksium.

Le langage PDDL a été développé originellement par Malik Ghallab et le comité responsable de la première compétition internationale de planification en 1998 (GHALLAB et al., 1998). Ce langage a été développé pour répondre aux besoins de standardisation de la communauté et permettre la comparaison des différents systèmes au travers de la diffusion et du partage de benchmarks. En contribuant à l'évaluation des systèmes et outils développés, PDDL a participé à l'amélioration continue des performances et de l'expressivité des planificateurs au cours des 20 dernières années. Depuis, PDDL est devenu un standard pour la description de domaines de planification.

Ce chapitre est organisé de la manière suivante : la première partie propose une rapide introduction au langage PDDL, la seconde donne les bases pour débuter avec PDDL4J, la troisième décrit l'architecture de PDDL4J ; finalement, la dernière partie propose une évaluation de PDDL4J avec les outils équivalents existants.

10.1 Une rapide introduction au langage PDDL

Le langage PDDL s'inspire des langages STRIPS (FIKES et NILSSON, 1971) et ADL (PEDNAULT, 1994). Il permet d'exprimer un problème de planification en dissociant ce qui est commun à plusieurs problèmes et ce qui est spécifique à chacun. La partie commune est décrite dans un fichier appelé domaine, qui contient la description des opérateurs de planification. La partie spécifique du problème, i.e., l'état initial et le but, est décrite dans un fichier appelé problème. Afin d'illustrer le langage PDDL sur lequel repose la librairie PDDL4J, nous proposons dans un premier temps d'introduire rapidement le langage PDDL au travers d'un exemple issu du domaine Logistics. Le domaine consiste à déplacer des objets par camion et par avion entre différentes villes.

1. http://pddl4j.imag.fr

10.1.1 La description du domaine de planification

Considérons tout d'abord le fichier domaine décrivant les opérateurs de planification.

```
(define (domain logistics)
  (:requirements :strips :equality :typing
                 :conditional-effects :universal-effects)
  (:types physobj - object
    obj vehicle - physobj
    truck airplane - vehicle
    location city - object
    airport - location)
  (:predicates (at ?x - physobj ?l - location)
        (in ?x - obj ?t - vehicle)
        (in-city ?l - location ?c - city))
  ...
```

La description d'un domaine débute toujours par la déclaration de son nom précédé du mot clé *domain*. La description se poursuit par la déclaration des prérequis du domaine. Les prérequis spécifient le niveau d'expressivité du domaine. Dans notre exemple, les prérequis spécifient que la description du domaine pourra utiliser des paramètres typés, des opérateurs pourront posséder des expressions d'égalité dans leurs préconditions, etc.

La plupart des domaines définissent ensuite les types qui seront utilisés dans la description des opérateurs. La définition des types est hiérarchique. En haut de la hiérarchie se trouve le type *object*, comme dans les langages orientés objets. Notons qu'il est possible d'exprimer de l'héritage multiple. Dans notre exemple, nous avons un type *vehicle* qui hérite du type *physobj*, qui hérite lui-même du type *object*.

Vient ensuite la déclaration des prédicats du domaine qui définissent le nom des prédicats avec leurs arguments typés qui pourront être utilisés dans les préconditions et les effets des opérateurs. Par exemple, le prédicat *(at ?x - phyobj ?l - location)* exprime qu'un objet *?x* de type *physobj* est situé dans un lieu *?l*.

Finalement, le domaine contient la description des opérateurs de planification qui définissent comment le monde évolue. Considérons l'opérateur de *load* de notre domaine Logistics :

```
(:action load
  :parameters (?o - object ?v - vehicle ?l - location)
  :precondition (and (at ?o ?l) (at ?v ?l))
  :effect (and (in ?o ?v) (not (at ?o ?l))))
```

L'opérateur exprime qu'un objet *?o* peut-être chargé dans un véhicule *?v* dans un lieu *?l* si et seulement si l'objet *o* et le véhicule *?v* sont situés au même lieu *?l*. L'exécution de l'action *load* aura pour effet de placer l'objet *?o* dans le véhicule *?v* et de le retirer du lieu *?l*.

De manière symétrique, l'opérateur *unload* :

```
(:action unload
  :parameters (?o - object ?v - vehicle ?l - location)
  :precondition (and (in ?o ?v) (at ?v ?l))
  :effect (and (not (in ?o ?v))))
```

exprime qu'un objet *?o* peut-être déchargé d'un véhicule *?v* dans un lieu *?l* si et seulement si l'objet *?o* est dans le véhicule *?v* et que ce véhicule est situé au lieu *?l*. L'exécution de l'action *unload* aura pour effet de retirer l'objet *?o* du véhicule *?v* et de le positionner au lieu *?l*.

Considérons maintenant un opérateur un peu plus compliqué possédant des effets conditionnels et une expression quantifiée universellement :

```
(:action fly
   :parameters (?o - object ?p - airplane ?from ?to - airport)
   :precondition (and (at ?p ?from) )
   :effect (and (at ?p ?to) (not (at ?p ?from))
                (forall (?o - object) (when (and (in ?o ?p))
                        (and (not (at ?o ?from)) (at ?p ?to)))))))
```

L'opérateur *fly* spécifie qu'un avion *?p* peut voler d'un aéroport de départ *?from* à un aéroport destination *?to* si et seulement si l'avion *?p* est situé initialement à l'aéroport *?from*. L'exécution de l'action *fly* aura pour effets de déplacer l'avion de la position *?from* à la position *?to*, mais également de déplacer tous les objets qu'il contient de la position *?from* à la position *?to*. Ce dernier effet ne sera effectif que si l'avion a été préalablement chargé.

Pour terminer la description de notre domaine, nous donnons ci-dessous l'opérateur *drive* :

```
(:action drive
  :parameters (?o - object ?t - truck ?from ?to - location ?c - city)
  :precondition (and (at ?t ?from)
                     (in-city ?from ?c)
                     (in-city ?t ?c))
  :effect (and (at ?t ?to) (not (at ?t ?from))
               (forall (?o - object) (when (and (in ?o ?t))
                       (and (not (at ?o ?from)) (at ?o ?to))))))
```

10.1.2 La description d'un problème de planification

Regardons maintenant comment exprimer un problème de planification simple pour le domaine défini précédemment :

```
(define (problem pb1)
   (:domain logistics)
   (:objects p1 p2 - obj
             london paris - city
             truck - truck
             plane - airplane
             north south east west - location
             lhr cdg  - airport)
   (:init (in-city cdg paris) (in-city lhr london)
          (in-city north paris) (in-city south paris)
          (at plane lhr) (at truck cdg)
          (at p1 lhr) (at p2 lhr))
   (:goal (and (at p1 north) (at p2 south))))
```

Le problème considéré (pb_1) décrit tout d'abord la liste des objets présents dans le monde. Dans notre exemple, le monde est constitué de deux objets p_1 et p_2, de deux villes *london* et *paris*, d'un camion *truck*, d'un avion *plane*, etc. Ensuite, le problème décrit l'état initial de ces objets, e.g., l'avion *plane* est situé à l'aéroport *lhr* tandis que le camion *truck* est situé à l'aéroport *cdg*. Finalement, le problème définit le but à atteindre. Dans notre exemple, le paquet p_1 doit être déposé dans le dépôt au nord de Paris et le paquet p_2 dans le dépôt au sud de Paris.

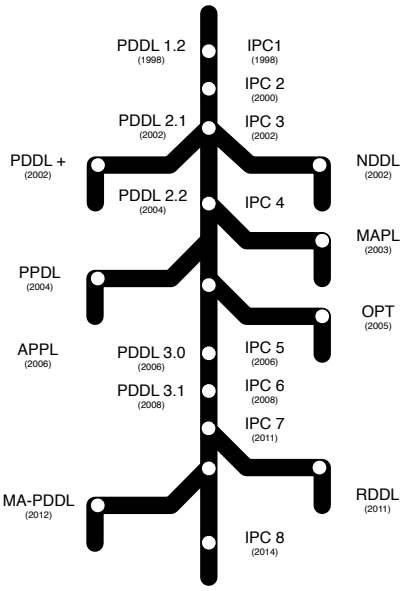


FIGURE 10.1 – Historique et évolutions du langage PDDL

10.1.3 Historique et évolutions du langage PDDL

La première version du langage PDDL1.2 est apparue en 1998 (GHALLAB et al., 1998). Cette version de PDDL a été le langage des deux premières compétitions internationales de planification IPC1 et IPC2 qui se sont tenues respectivement en 1998 et 2000. Une vue globale des évolutions et des extensions du langage PDDL est représentée à la figure 10.1.

En 2002, une nouvelle version PDDL2.1 (FOX et LONG, 2003) est proposée et devient la nouvelle référence pour la troisième édition d'IPC. Cette version de PDDL introduit notamment les fonctions numériques, la dimension temporelle des opérateurs de planification ou encore la notion de métrique pour les plans solutions. Toutes ces extensions visent à modéliser des problèmes de plus en plus proches des problèmes réels. Parallèlement, trois extensions de PDDL2.1 ont vu le jour. La première extension (FOX et LONG, 2002, 2006), appelée PDDL+, étend PDDL2.1 pour permettre de modéliser les événements exogènes et les effets continus. La seconde extension (BRENNER, 2003), appelée MAPL (Multi-Agent Planning Langage), introduit dans le langage la possibilité d'exprimer des éléments de synchronisation entre actions, mais également des actions de communication entre agents. La dernière extension, appelée OPT [2] (*Onthology with Polymorphic Types*) propose une notation permettant l'intégration du langage PDDL avec des ontologies. Finalement, à la même période la NASA a proposé un langage appelé NDDL (FRANK et JÓNSSON, 2003) reposant sur un formalisme FDR (Finit Domain Representation) pour exprimer des problèmes de planification.

En 2004, une révision mineure du langage est proposée pour la 4e compétition internationale de planification IPC4. Cette version du langage PDDL2.2 (EDELKAMP et HOFFMANN, 2004) introduit l'inférence logique et permet dorénavant de spécifier des prédicats temporels. Au même instant apparaît une version probabiliste de PDDL appelée PPDDL (*Probabilistic PDDL*) (YOUNES et LITTMAN, 2004). Cette extension sera utilisée pour les *tracks* probabilistes des compétitions IPC4 et IPC5.

2. MCDERMOTT, 2005

En 2006, une nouvelle évolution PDDL3.0 est apportée au langage pour la 5e édition d'IPC (GEREVINI et LONG, 2005a,b,c). Il est dorénavant possible d'exprimer des contraintes sur le plan solution recherché. Ces contraintes peuvent être obligatoirement respectées ou de simples préférences à satisfaire. Concomitamment, une nouvelle version du langage NDDL est proposée (BUTLER et MUÑOZ, 2006). Cette version du langage s'appelle APPL (*Abstract Plan Preparation Language*).

Finalement, la dernière version de PDDL, la version 3.1 (HELMERT, 2008) a été proposée en 2008. Cette version du langage a été utilisée pour les compétitions IPC6, IPC7 et IPC8. Il est possible de manipuler des termes composés. Deux extensions de PDDL3.1 doivent être mentionnées pour terminer. La première (SANNER, 2010) est le langage RDDL (*Relational Dynamic influence Diagram Language*) qui est une extension de PDDL3.1 pour la planification probabiliste. Ce langage a servi pour la *track* probabiliste d'IPC7. La dernière extension (KOVACS, 2012) a donné lieu au langage MA-PDDL (*Multi-Agent PDDL*) pour la spécification de problèmes multiagents de planification.

10.2 Débuter avec PDDL4J

10.2.1 Installation, compilation et exécution

Débuter avec PDDL4J est simple. La ligne de commande ci-dessous montre comment procéder :

```
1 $ git init
2 $ git clone https://github.com/pellierd/pddl4j.git
3 $ cd pddl4j
4 $ ./gradlew clean build
5 $ ./gradlew javadoc
6 $ ./gradlew run -PArgs=-o,<DOMAIN FILE PATH>,-f,<PROBLEM FILE PATH>
```

Le système de gestion de versions GIT permet d'installer simplement PDDL4J (ligne 1 et 2). PDD4L est installée avec tous les fichiers de configuration nécessaires à la compilation avec l'outil Gradle (ligne 4 et 5). Ces lignes compilent PDDL4J et génèrent la documentation interne. Des exemples de domaines et de problèmes sont disponibles dans le répertoire `./pddl` à la racine du projet pour tester le planificateur par défaut de la librairie HSP (*Heuristic Search Planner*) (ligne 6). Les plans générés par PDDL4J sont compatibles avec PDDL3.1 et peuvent être validés avec VAL, un validateur de plans utilisé pour les compétitions de planification et développé par (HOWEY, LONG et FOX, 2004)

Pour lancer directement le planificateur par défaut HSP sans utiliser l'outil Gradle, il suffit de lancer la commande suivante :

```
> java -javaagent:build/libs/pddl4j-VERSION.jar -server -Xms2048m -
    Xmx2048m fr.uga.pddl4j.planners.hsp.HSP -o <DOMAIN FILE PATH> -f <
    PROBLEM FILE PATH>
```

Notons que Gradle génère le jar exécutable pour la machine virtuelle Java dans le répertoire `/build/libs`.

10.2.2 Utilisation en tant que code tiers

Le code ci-dessous montre comment faire appel au planificateur par défaut HSP directement dans une classe Java :

```
1 public static void main(String[] args) {
```

```
2
3       // Get the domain and the problem from the command line
4       String domain = args[0];
5       String problem = args[1];
6
7       // Create the problem factory
8       final ProblemFactory factory = new ProblemFactory();
9
10      // Parse the domain and the problem
11      ErrorManager errorManager = factory.parse(domain, problem);
12      if (!errorManager.isEmpty()) {
13            errorManager.printAll();
14            System.exit(0);
15      }
16
17      // Encode and simplify the planning problem in a compact
    representation
18      final CodedProblem pb = factory.encode();
19      if (!pb.isSolvable()) {
20            System.out.println("goal can be simplified to FALSE. "
21                        + "no search will solve it"));
22            System.exit(0);
23      }
24
25      // Create the planner and choose the Fast Forward heuristic
26      HSP planner = new HSP();
27      planner.setHeuristicType(Heuristic.Type.FAST_FORWARD);
28
29      // Search for a solution plan
30      final Plan plan = planner.search(pb);
31      if (plan != null) {
32            System.out.println(String.format("%nfound plan as follows:%n%
    n"));
33            System.out.println(pb.toString(plan));
34      } else {
35            strb.append(String.format("%nno plan found%n%n"));
36      }
37  }
```

Tout d'abord, nous récupérons les chemins correspondant aux fichiers *problème* et *domaine* (lignes 4 et 5). Puis nous créons la fabrique de problèmes (ligne 8). L'objet *fabrique* permet d'effectuer l'analyse lexicale et syntaxique des fichiers ainsi que l'édition de lien des fichiers PDDL (ligne 11). Lorsque l'analyse est terminée, la fabrique instancie et encode le problème de planification de manière compacte (ligne 18). La méthode *isSolvable()* teste en temps polynomial si le problème est résoluble (ligne 19) ou non avant de lancer la recherche d'un plan solution. Un objet planificateur est créé ensuite (ligne 26). Dans notre exemple, nous créons une instance du planificateur par défaut HSP (cf. §10.3.6 pour connaître la liste des planificateurs disponibles dans PDDL4J). Puis, nous choisissons l'heuristique à utiliser (ligne 27). Dans notre exemple, l'heuristique du planificateur FastForward (HOFFMANN et NEBEL, 2001) est choisie (cf. §10.3.5 pour connaître la liste des heuristiques disponibles dans PDDL4J). Finalement, la recherche d'un plan solution est lancée (ligne 30) et le plan solution est affiché, s'il existe.

10.3 L'architecture de PDDL4J

PDDL4J est composé de plusieurs modules indépendants (cf. figure 10.2). PDDL4J comprend un analyseur syntaxique compatible PDDL3.1 validé sur tous

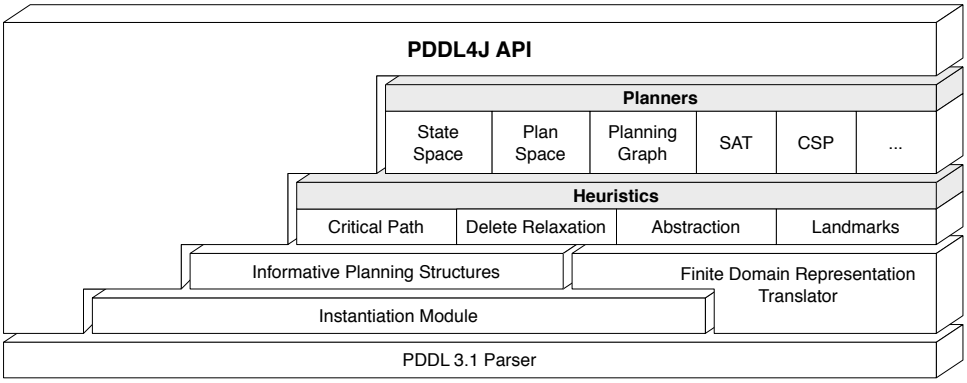


FIGURE 10.2 – Architecture de la librairie PDDL4J

les benchmarks classiques des compétitions internationales de planification IPC. Au-dessus de l'analyseur, PDDL4J fournit un module de prétraitement, le module d'instanciation. Ce module instancie de manière efficace les actions des problèmes de planification tout en simplifiant, lorsque c'est possible, leur représentation. Puis nous trouvons deux modules : un premier module qui regroupe un certain nombre de structures très utilisées en planification, et sur lesquelles sont construites beaucoup d'heuristique pour la planification, et un second module qui encode les problèmes de planification au format FDR (*Finit Domain Representation*). Finalement, PDDL4J comprend deux modules de haut niveau : l'un qui implémente les principales heuristiques développées en la planification, et l'autre qui implémente un certain nombre de planificateurs classiques.

10.3.1 L'analyseur syntaxique PDDL

L'analyseur syntaxique PDDL de PDDL4J respecte la syntaxe PDDL3.1 (KOVACS, 2012). Ce module assure l'analyse lexicale, syntaxique et sémantique des fichiers contenant la description des domaines et des problèmes de planification. De manière identique à un analyseur de compilateur, l'analyseur de PDDL4J transforme les fichiers en entrée en une représentation compacte facilitant la génération de code exécutable. Dans notre cas particulier, il construit plusieurs tables de symboles à partir des symboles de constantes, de prédicats, de fonctions, d'opérateurs, etc., contenus dans le domaine et dans le problème et encode les expressions sous la forme d'arbre d'entiers pouvant être traités de manière efficace par le module d'instanciation (cf. figure 10.3).

L'analyseur de PDDL4J est développé avec l'outil JavaCC (*Java Compiler Compiler*[3]). JavaCC est un générateur d'analyseur syntaxique. JavaCC prend en entrée un fichier de description du langage PDDL proche de la BNF (*Backus-Naur Form)*) décrivant les règles syntaxiques du langage PDDL, et produit en sortie l'analyseur syntaxique du langage PDDL en Java. JavaCC est bien adapté à PDDL4J parce qu'il permet de gérer et de faire évoluer l'analyseur rapidement pour tenir compte des évolutions du langage PDDL. De plus, JavaCC possède un bon système pour identifier les erreurs facilitant le débogage des domaines et des problèmes de planification.

L'analyse sémantique effectuée par l'analyseur détecte un certain nombre de situations non critiques, mais pouvant conduire à des erreurs :

3. https://javacc.org

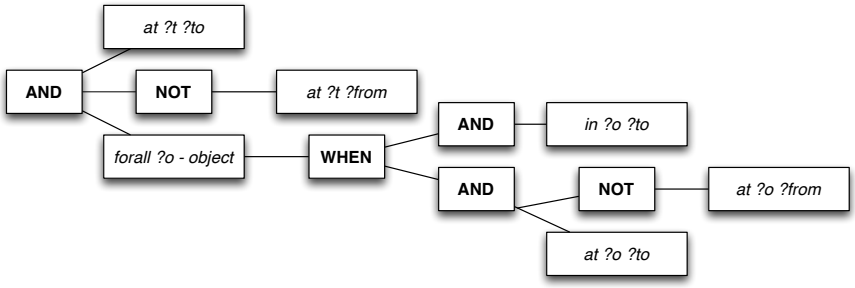


FIGURE 10.3 – Exemple de représentation des effets de l'opérateur *drive* du domaine Logistics

— une constante, une variable quantifiée, un prédicat, une fonction ou encore un type déclaré n'est jamais utilisé dans le domaine ou le problème de planification ;
— un paramètre déclaré dans un opérateur n'est jamais utilisé dans sa description ;
— le nom du domaine défini dans le fichier domaine est différent du nom de domaine déclaré dans le fichier de problème ;
— etc.

L'analyseur détecte également des erreurs critiques :
— un type, un prédicat, une constante, ou une fonction est utilisé sans être déclaré ;
— la hiérarchie de types est incohérente ;
— une variable quantifiée n'est pas typée ;
— etc.

10.3.2 Le module d'instanciation

Le rôle du module d'instanciation est de dénombrer les actions du problème de planification à partir des opérateurs définis dans le domaine. Cette étape préalable à la recherche d'un plan solution est réalisée par la plupart des planificateurs modernes. C'est notamment une étape préalable nécessaire aux planificateurs reposant sur des solveurs SAT (KAUTZ et SELMAN, 1992, 1999 ; RINTANEN, 2012) et CSP (BARTÁK, SALIDO et ROSSI, 2010 ; KAMBHAMPATI, 2000 ; LOPEZ et BACCHUS, 2003). D'autres planificateurs utilisent également le résultat de l'instanciation pour calculer efficacement des heuristiques pour guider la recherche (HOFFMANN et NEBEL, 2001) ou simplifier et réduire la complexité des problèmes de planification (HELMERT et DOMSHLAK, 2009).

L'implémentation du module d'instanciation repose sur les travaux de (KOEHLER et HOFFMANN, 1999) développés initialement pour le planificateur IPP (KOEHLER et al., 1997). Étant donné un opérateur, le processus d'instanciation consiste à remplacer tous les paramètres typés, quantifiés ou non, par les constantes du même type déclarées dans le domaine et le problème. Ce processus, comme nous l'avons évoqué au chapitre 9, est coûteux, mais peut être optimisé tout en simplifiant les problèmes de planification en éliminant les actions qui ne pourront jamais être déclenchées.

Le processus complet d'instanciation est composé de 6 étapes :

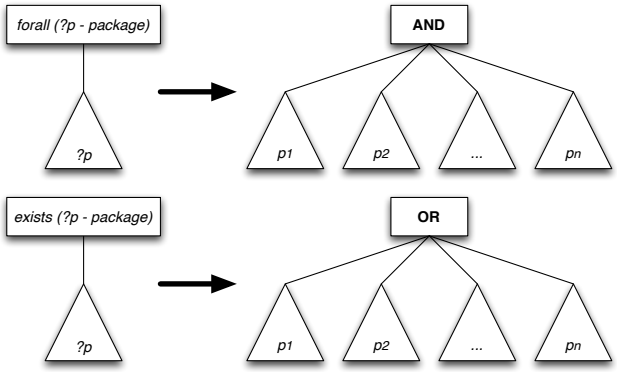


FIGURE 10.4 – Suppression des expressions quantifiées par énumération

Étape. 1 (Normalisation des expressions logique)

Toutes les expressions de la forme $\psi \rightarrow \varphi$ sont remplacées par leur forme équivalente $\neg\psi \vee \varphi$. Les négations sont déplacées de telle manière à porter sur les formules atomiques du langage, e.g., les expressions de la forme $\neg(\psi \wedge \varphi)$ sont remplacées par $\neg\psi \vee \neg\varphi$. Finalement, toutes les expressions quantifiées sont supprimées et remplacées par leur expression équivalente (cf. figure 10.4).

Étape. 2 (Inférence des types)

Cette seconde étape est optionnelle. Elle concerne les domaines de planification non typés et consiste à inférer les types des paramètres des opérateurs préalablement à l'étape d'instanciation. L'inférence s'appuie sur les prédicats unaires déclarés dans le domaine. L'inférence de types réduit le domaine de valeurs des paramètres des opérateurs, ce qui a pour conséquence de réduire le nombre d'instanciations possibles et d'accélérer de manière importante le processus d'instanciation des domaines non typés.

Étape. 3 (Instanciation des opérateurs)

Au cours de cette étape, chaque paramètre d'un opérateur est remplacé par une constante du même type. La valeur affectée au paramètre est ensuite propagée dans les préconditions et les effets de l'opérateur. Pour chaque nouvelle affectation, une nouvelle structure arborescente représentant l'opérateur est créée. Lors que le processus d'instanciation se termine, tous les paramètres ont une valeur. On parle alors d'opérateur instancié ou d'action. Le processus d'instanciation s'applique pour chaque opérateur du domaine.

Étape. 4 (Simplification des préconditions et des effets des actions)

Cette étape consiste à simplifier les expressions logiques représentant les préconditions et les effets des actions. La valeur de vérité de certaines propositions peut être déterminée *a priori*. Ces simplifications reposent sur le principe d'inertie dont nous avons déjà parlé (cf. §9.3.2).

Étape. 5 (Simplification des actions)

Cette étape consiste à généraliser la simplification effectuée sur les préconditions et les effets des actions aux actions elles-mêmes. Supposons que l'expression logique complète représentant les préconditions d'une action puisse être simplifiée à faux. Cela signifie que l'action ne pourra jamais s'appliquer. Dans ce cas, l'action peut être tout simplement supprimée du problème de planification. Par extension, cette simplification s'applique également aux actions dont les effets peuvent être simplifiés à faux.

Étape. 6 (Encodage compact des actions avec des vecteurs de bits)

La dernière étape consiste à encoder les actions de manière compacte afin d'optimiser l'utilisation de la mémoire et les tests de déclenchement des actions pendant la recherche d'un plan solution. Le principe de l'encodage est de représenter les préconditions ou les effets d'une action avec deux vecteurs de bits : l'un représentant les propositions positives et l'autre les propositions négatives. Pour une description plus détaillée du processus d'instanciation, nous invitons le lecteur à se reporter à (PELLIER et FIORINO, 2017).

10.3.3 L'encodage FDR

La représentation logique des problèmes de planification oblige à considérer des états qui ne sont pas nécessairement pertinents pour la recherche d'un plan solution. Il existe une autre représentation classique qui pallie en partie ce problème. Cette représentation est appelée FDR (*Finit Domain Representation*). Supposons que nous devions encoder une proposition *(at p1 east)* exprimant que le paquet p_1 est situé à l'est. Supposons également qu'un paquet est forcément positionné sur un unique point cardinal à un instant donné, i.e., *west, east, south* ou *north*. Il est possible d'encoder la proposition *(at* p_1*)* par une seule et unique variable dont le domaine est $\{$*west, east, south, north*$\}$. La représentation est ici plus compacte puisque les états incohérents où le paquet p_1 occupe plusieurs positions à un instant donné sont éliminés de la représentation du problème. Toutefois, encore faut-il être capable de déterminer à partir du domaine PDDL qu'un paquet ne peut pas occuper plusieurs positions à un même instant. Autrement dit, il faut être capable de calculer des *invariants* à partir du domaine pour être en mesure d'exploiter le caractère compact de la représentation FDR.

La librairie PDDL4J propose une implémentation des mécanismes d'encodage des problèmes de planification dans une représentation FDR. Cette implémentation repose sur les techniques décrites dans (HELMERT et DOMSHLAK, 2009). L'encodage prend en entrée un problème exprimé dans le langage PDDL et produit un problème dans la représentation FDR.

Définition 10.1 (Un problème de planification)**.** *Un problème de planification FDR est un quintuplet* (V, A, c, I, G) *où :*

- V *est un ensemble de variables. Chaque variable* $v \in V$ *possède un domaine fini* $Dom(v)$ *;*
- A *est un ensemble d'actions ;*
- $c : A \rightarrow \mathbb{R}^+$ *est une fonction de coût ;*
- I *est une affectation complète des variables de* V *représentant l'état initial du problème ;*
- G *est une affectation partielle des variables de* V *représentant le but à atteindre.*

Définition 10.2 (Action). *Une action est un triplet* $a = (name(a), precond(a), effect(a))$ *où* $name(a)$ *est le nom de l'action,* $precond(a)$ *et* $effects(a)$ *sont des affectations de la forme* $(v = d)$ *telles que* v *est une variable appartenant respectivement aux préconditions et aux effets de* a *et* $d \in Dom(v)$*. Finalement, un état* s *est une affectation complète des variables du problème. Une action* a *est applicable dans un état* s *si et seulement si* $precond(a) \subseteq s$*.*

De manière similaire à la représentation logique, une solution à un problème de planification FDR peut être définie comme suit :

Définition 10.3 (Plan solution). *Une solution à un problème de planification* (V, A, c, I, G) *est un plan* π *tel que* $G \subseteq \gamma(I, \pi)$*.*

L'encodage est réalisé en 3 étapes :

Étape. 1 (Sélection des variables)

L'étape de sélection des variables consiste à choisir les variables qui encodent le plus de propositions possible en calculant des invariants du problème de planification. La procédure implémentée dans PDDL4J ne garantit pas de trouver la représentation la plus compacte. Ce problème est un problème NP-Difficile, mais s'appuie sur le meilleur algorithme connu approximant la solution (AUSIELLO et al., 1999).

Étape. 2 (Encodage de l'état initial et du but)

Cette étape consiste à encoder l'état initial et le but à partir des variables identifiées à l'étape 1.

Étape. 3 (Encodage des actions)

De manière comparable à l'étape précédente, cette étape consiste à encoder les préconditions et les effets des actions à partir des variables identifiées à l'étape 1.

10.3.4 Les structures informatiques de données

Beaucoup de travaux en planification s'appuient sur l'utilisation des mêmes structures de données. La plupart de celles-ci ont été inventées afin d'encoder l'espace de recherche de manière compacte et de faciliter l'extraction d'information à partir des problèmes de planification pour l'élaboration d'heuristiques informatives. Le graphe de planification introduit au chapitre 5 est l'une de ces structures, mais ce n'est pas la seule. La librairie PDDL4J implémente trois des principales structures de données : le graphe de planification, le graphe de causalité et le graphe de transition de domaine. Ces structures peuvent être utilisées directement pour développer de nouveaux mécanismes de planification comme nous l'avons fait au chapitre 5 et sont utilisées pour le calcul de certaines heuristiques (cf. §10.3.5) et pour certains planificateurs (cf. §10.3.6) implémentés dans PDDL4J.

Le graphe de planification

Le *graphe de planification* en tant que structure de données a été proposé pour la première fois par (BLUM et FURST, 1997). Pour la définition formelle de cette structure et de ces propriétés, nous invitons le lecteur à se reporter au chapitre 5. Le graphe de planification du problème Logistics présenté en début de chapitre est

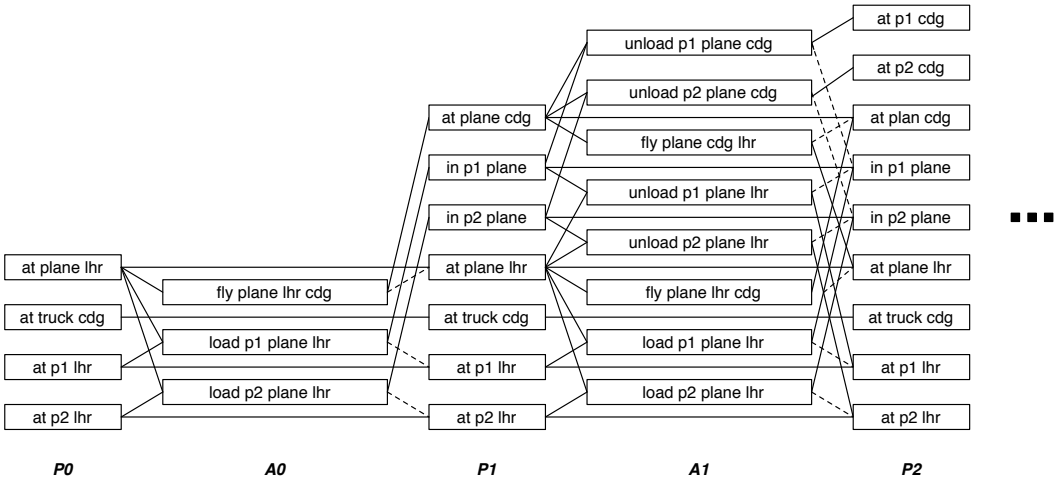


FIGURE 10.5 – Les deux premiers niveaux du graphe de planification du problème Logistics défini §10.1.2. Chaque boîte représente une action ou une proposition. Les lignes pleines représentent les préconditions des actions et les lignes pointillées leurs effets. Pour des raisons de simplicité, les relations d'exclusion ne sont pas représentées.

donné à la figure 10.5. L'implémentation dans la librairie PDDL4J de cette structure de données repose sur celle proposée par (KOEHLER et HOFFMANN, 1999). Notons que ce n'est pas la seule implémentation documentée existante, e.g., (KAUTZ et SELMAN, 1999 ; LONG et FOX, 1999).

Le graphe de causalité

Le concept de *graphe de causalité* a été introduit pour la première fois par (KNOBLOCK, 1994). Un graphe de causalité exprime les relations de dépendance entre les variables d'un problème de planification utilisant une représentation FDR. Cette structure a notamment été utilisée pour des sous-classes de problèmes de planification facile à résoudre (BRAFMAN et DOMSHLAK, 2003 ; JONSSON et BÄCKSTRÖM, 1998 ; WILLIAMS et NAYAK, 1997), pour décomposer un problème de planification en sous-problèmes indépendants (BRAFMAN et DOMSHLAK, 2006 ; JONSSON, 2007 ; KNOBLOCK, 1994), pour développer des heuristiques pour guider la recherche d'un plan solution (HELMERT, 2006), ou encore pour l'analyse de la topologie des espaces de recherche des benchmarks de planification (HOFFMANN, 2011).

Plus formellement, un graphe de causalité est un graphe orienté dont les nœuds sont les variables du problème de planification. Les arcs du graphe expriment les dépendances entre les variables. Il existe un arc entre deux nœuds u et v s'il existe une action a telle que v est une précondition de a et u un effet de a ou si v et u sont des effets de a. La figure 10.6 représente un graphe de causalité typique du domaine Logistics. Nous pouvons observer que les paquets p_1 et p_2 sont indépendants. En revanche, déplacer le camion ou l'avion modifie l'état des paquets p_1 et p_2. Notons que contrairement au graphe de planification, le graphe de causalité n'exprime que les dépendances entre les actions du problème de planification, indépendamment de l'état initial et du but du problème.

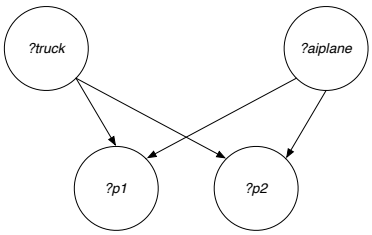


FIGURE 10.6 – Exemple de graphe de causalité typique d'un problème issu du domaine Logistics : les nœuds représentent les variables du problème encodé au format FDR et les arcs les relations de dépendance entre actions.

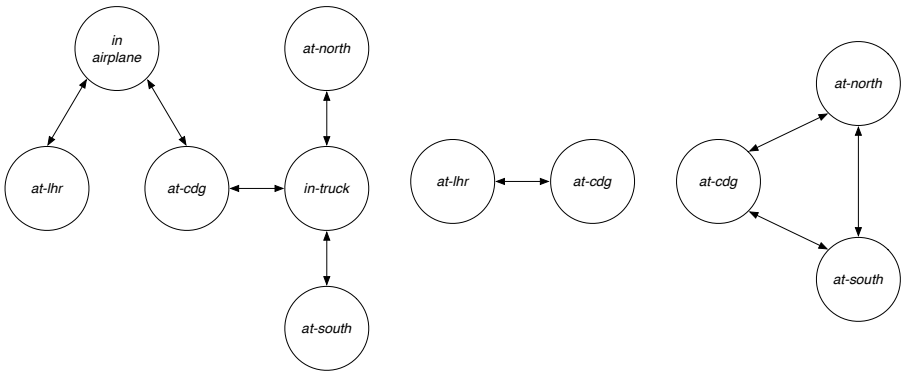


FIGURE 10.7 – Graphes de transitions de domaine extrait d'un problème du domaine Logistics pour les variables p_1 et p_2 (left), la variable *airplane* (centre) et la variable *?truck* (right) : les nœuds sont les valeurs des variables et les arcs les transitions possibles. Pour des raisons de simplicité, les actions ne sont pas indiquées sur les arcs.

Le graphe de transitions de domaine

La dernière structure informatique implémentée dans la librairie PDDL4J et souvent utilisée conjointement avec le graphe de causalité est le *graphe de transitions de domaine*. Cette structure décrit comment une variable d'un problème de planification reposant sur une représentation FDR évolue en fonction des actions du problème. Un graphe de transitions de domaine est un graphe orienté dont les nœuds sont les valeurs du domaine de la variable considérée. Les arcs du graphe sont étiquetés par les actions du problème. Il existe un arc entre deux nœuds d et d' s'il existe une action a telle que $(v = d) \in \text{precond}(a)$ et $(v = d') \in \text{effects}(a)$ ou $v \notin \text{precond}(a)$ et $(v = d') \in \text{effects}(a)$. Un graphe de transitions de domaine est dit *inversible* si pour chaque arc (d, d'), il existe un arc (d', d). Un exemple de graphe de transitions de domaine extrait du domaine Logistics est donné à la figure 10.7. Il a été prouvé (CHEN et GIMÉNEZ, 2010) que l'existence d'une solution pour un problème de planification possédant un graphe de causalité acyclique et un graphe de transitions de domaine inversible peut être calculé en temps polynomial. En revanche, si le graphe de causalité d'un problème de planification contient des cycles alors calculer l'existence d'un plan solution est NP-difficile.

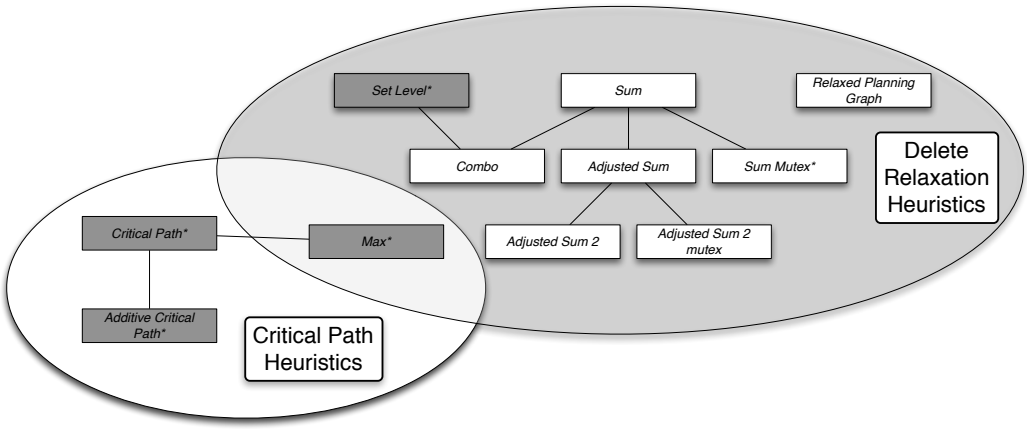


FIGURE 10.8 – Vue globale des grandes familles d'heuristiques en planification et de leur implémentation dans la bibliothèque PDDL4J. Les heuristiques marquées d'une étoile * sont admissibles. Les liens entre les heuristiques expriment leur lien de parenté.

10.3.5 Les heuristiques

Les compétitions internationales de planification sont largement dominées par les planificateurs s'appuyant sur des algorithmes de recherche heuristique, en particulier pour résoudre des problèmes de planification déterministes. Les méthodes de recherche heuristique sont maintenant devenues un standard (BRYCE et KAMBHAMPATI, 2007 ; HASLUM et al., 2007 ; LIANG, 2012) dans la communauté pour explorer de grands espaces de recherche. Ces méthodes ont permis à la planification de faire un saut d'échelle dans la complexité des problèmes traités. Le principe de ces approches est de guider la recherche par une fonction heuristique h qui estime la distance de l'état courant s en termes d'actions à un état contenant le but g. Pratiquement, la fonction heuristique permet de choisir le meilleur état encore non exploré, i.e., celui dont la valeur $h(s, g)$ est la plus petite pour atteindre le but. Une heuristique est dite admissible si elle ne surestime jamais la distance réelle pour atteindre un état contenant le but. Dans ce cas, il a été prouvé que le premier plan solution retourné par l'algorithme A* est optimal.

Toute la difficulté est de trouver des heuristiques qui sont à la fois facilement calculables et très informatives et qui peuvent être dérivées à partir des représentations classiques utilisées en planification. Une méthode largement utilisée pour concevoir de nouvelles heuristiques consiste à chercher à résoudre un problème de planification *relaxé*, i.e., plus simple et pouvant être résolu en temps polynomial. En l'état (cf. figure 10.8), la librairie PDDL4J implémente des heuristiques fondées sur le calcul d'un chemin critique, e.g., (HASLUM, BONET et GEFFNER, 2005) et des heuristiques fondées sur la résolution d'un problème relaxé, e.g., (DOMSHLAK, HOFFMANN et KATZ, 2015). Les heuristiques marquées d'une étoile * sont admissibles. Pour plus de détails sur les heuristiques et leur implémentation dans PDDL4J, nous invitons le lecteur à se référer à (PELLIER et FIORINO, 2017). La plupart de leur implémentation repose sur les structures de données décrites au paragraphe précédent.

10.3.6 Les planificateurs

La couche de plus haut niveau de la librairie PDDL4J implémente un certain nombre de planificateurs. L'objectif n'est pas d'être exhaustif, mais plutôt d'être pédagogique en proposant des implémentations des principaux planificateurs tout

en simplifiant l'évaluation de nouveaux planificateurs avec l'état de l'art. Pour l'instant, les planificateurs implémentés dans PDDL4J peuvent être regroupés en trois catégories :

Les planificateurs fondés sur la recherche dans un espace d'états

Dans cette catégorie, PDDL4J propose une implémentation des planificateurs suivants :

— **HSP (***Heuristic Search Planner***)** (BONET et GEFFNER, 2001) repose sur une recherche en chaînage avant couplée à une heuristique de type chemin critique. La version implémentée dans PDDL4J a été augmentée. Elle permet de choisir la stratégie de recherche à utiliser, i.e., profondeur d'abord, largeur d'abord, profondeur itérative, A*, etc., et l'heuristique pour guider la recherche (cf. section précédente pour connaître la liste des heuristiques disponibles).

— **FastForward** (HOFFMANN et NEBEL, 2001) repose sur une recherche de type descente de gradient couplée à une heuristique fondée sur l'extraction d'un plan solution à partir d'un graphe de planification relaxé.

— **FastDownard**(HELMERT, 2006) est un planificateur qui s'appuie sur la représentation FDR des problèmes de planification. La stratégie de recherche est de type recherche en profondeur d'abord. Le planificateur exploite le graphe de causalité pour filtrer les actions les plus prometteuses à exécuter dans un état donné.

Les planificateurs fondés sur la recherche dans un graphe de planification

Contrairement aux planificateurs précédents, la recherche d'un plan solution est réalisée en explorant le graphe de planification construit à partir du problème de planification. Ce type de planificateurs s'articule par conséquent autour de deux étapes : une étape de construction du graphe de planification et une étape de recherche. La librairie PDDL4J implémente trois planificateurs de cette catégorie qui se différencient par la stratégie de recherche utilisée pour extraire un plan solution à partir du graphe de planification :

— **Graphplan** (BLUM et FURST, 1997) est le planificateur qui a introduit pour la première fois le concept de graphe de planification. La recherche d'un plan solution est effectuée par chaînage arrière directement à partir du graphe de planification.

— **SATPlan** est une implémentation du planificateur proposé par (KAUTZ et SELMAN, 1999; KAUTZ, SELMAN et HOFFMANN, 2006). La recherche est réalisée par encodage du graphe de planification sous la forme d'un problème SAT. La résolution du problème SAT est réalisée avec la librairie SAT4J (LE BERRE et PARRAIN, 2010).

— **GP-CSP** est une implémentation du planificateur proposé par (KAMBHAMPATI, 2000). Le graphe de planification est encodé cette fois sous la forme d'un problème CSP. L'implémentation de PDDL4J utilise la librairie Choco (PRUD'HOMME, FAGES et LORCA, 2014) pour effectuer la recherche d'un plan solution.

Les planificateurs fondés sur des solveurs SAT

L'idée générale sous-jacente à cette catégorie de planificateurs est d'encoder le problème de planification sous la forme d'un problème SAT afin de bénéficier de

l'efficacité des solveurs développés dans cette communauté. Ces dernières années ont vu une augmentation importante des performances des solveurs SAT, notamment ceux capables d'exploiter les architectures CPU multicœurs (RINTANEN, HELJANKO et NIEMELÄ, 2006). Parallèlement, des avancées importantes pour encoder de manière compacte les problèmes de planification sous la forme de problèmes SAT couplées au développement de nouvelles heuristiques ont été réalisées (RINTANEN, 2012). La littérature distingue trois types d'encodage qui permettent d'obtenir des plans d'actions plus ou moins concurrents. Le premier type d'encodage ne permet d'obtenir que des plans séquentiels (KAUTZ et SELMAN, 1992). Le second type d'encodage permet d'obtenir des plans définis comme des séquences d'ensembles d'actions concurrentes (RINTANEN, HELJANKO et NIEMELÄ, 2006 ; ROBINSON et al., 2009). Finalement, le troisième encodage (RINTANEN, 2014) permet d'obtenir des plans encore plus expressifs en termes de concurrence, proches de ceux générés par les planificateurs fondés sur une recherche dans un espace de plans tel que UCPOP (PENBERTHY et WELD, 1992). PDDL4J implémente un planificateur appelé PSPlan qui s'appuie sur plusieurs de ces encodages. La résolution des problèmes est effectuée grâce solveurs développés dans la librairie SAT4J (LE BERRE et PARRAIN, 2010).

10.4 Comparaison et évaluation

Il est difficile de proposer une évaluation complète de tous les modules et de tous les planificateurs développés dans PDDL4J avec leur implémentation originelle. Certains codes ne sont tout simplement plus disponibles ou utilisent des librairies qui ne le sont plus. Nous proposons toutefois dans cette section une évaluation de l'analyseur syntaxique et du module d'encodage et d'instanciation implémentés dans PDDL4J avec les modules équivalents des planificateurs FastDownward (FD) (HELMERT, 2006) et FastForward (FF) (HOFFMANN et NEBEL, 2001) qui ont été intégrés dans la boîte à outils LAPT (*Ligtweight Automated Planning ToolKit*) maintenue par (RAMIREZ, LIPOVETZKY et MUISE, 2015). FastDownward a été principalement implémenté avec le langage Python et FastForward en langage C. Les expériences ont été réalisées sur une machine équipée d'un processeur Intel Xeon 6 cœurs cadencés à 2.2Ghz. Nous avons effectué les tests sur les 29 domaines de planifications déterministes élaborés pour les différentes compétitions internationales de planification depuis 2000. Pour chaque domaine, nous avons comparé en moyenne pour tous les problèmes du domaine (1) le temps nécessaire pour effectuer l'analyse lexicale, syntaxique et sémantique (cf. figure 10.9) (2) le temps nécessaire à l'instanciation et à l'encodage (cf. figure 10.10) et (3) la taille moyenne de la mémoire nécessaire à leur stockage (cf. figure 10.11).

Les résultats montrent que l'analyseur implémenté dans PDDL4J est relativement performant par rapport à l'analyseur de FD écrit en python et à celui de FF développé en C. Il surpasse notamment toujours celui de FD. Le principal avantage de l'analyseur de PDDL4J est qu'il respecte la syntaxe complète de PDDL3.1, ce qui n'est pas le cas de FD et FF.

En ce qui concerne le temps d'instanciation et d'encodage, la librairie est également très compétitive. FD est largement surpassé par PDDL4J et PDDL4J est légèrement surclassée par FF sur 27 des 29 domaines. Les deux domaines où FD surpasse FF et PDDL4J sont des domaines possédant des opérateurs avec un nombre important de paramètres. En effet, les implémentations respectives de FF

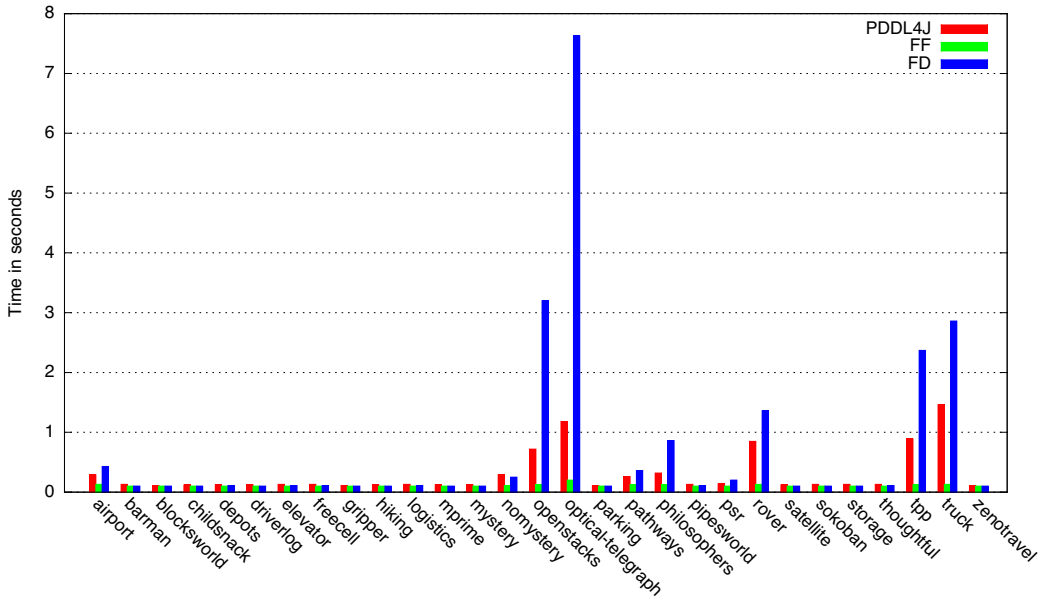


FIGURE 10.9 – Temps moyen pour effectuer l'analyse lexicale, syntaxique et sémantique des fichiers de domaines et de problèmes

et de PDDL4J reposent sur un mécanisme de simplification très coûteux lorsque les opérateurs du domaine ont un nombre important de paramètres.

En ce qui concerne la taille mémoire nécessaire au stockage des problèmes, PDDL4J, FD et FF sont relativement comparables.

Pour résumer, les résultats montrent que PDDL4J est compétitive par rapport aux implémentations de FD et FF, et ce malgré le choix du langage Java. Le principal avantage de PDDL4J repose sur la maturité de son code. En effet, contrairement aux deux autres implémentations qui sont des codes de recherche modifiés de manière incrémentale, PDDL4J est un code développé en respectant dès la conception une démarche de génie logiciel et les standards de développement. PDDL4J intègre notamment des outils pour veiller à sa non-régression, tels que SonarCube [4], et à son intégration continue tels que Jenkins [5]. Notons finalement que PDDL4J est disponible avec une documentation complète de son code sous licence LGPL.

10.5 Conclusion

La librairie PDDL4J est une boîte à outils mature et *open source* dont l'objectif est de faciliter l'intégration de techniques issues de la planification automatique dans d'autres communautés, mais également de faciliter au sein de la communauté de planification le développement de nouveaux planificateurs. PDDL4J fournit les briques de base pour manipuler le langage PDDL, une API Java documentée pour développer de nouveaux algorithmes de planification, des implémentations des principales heuristiques de planification et un certain nombre de planificateurs à l'état de l'art utilisables directement dans différents cas applicatifs.
Les futurs développements et extensions de PDDL4L porteront sur les points suivants :

4. http://pddl4j.imag.fr/sonar/
5. http://pddl4j.imag.fr/jenkins/

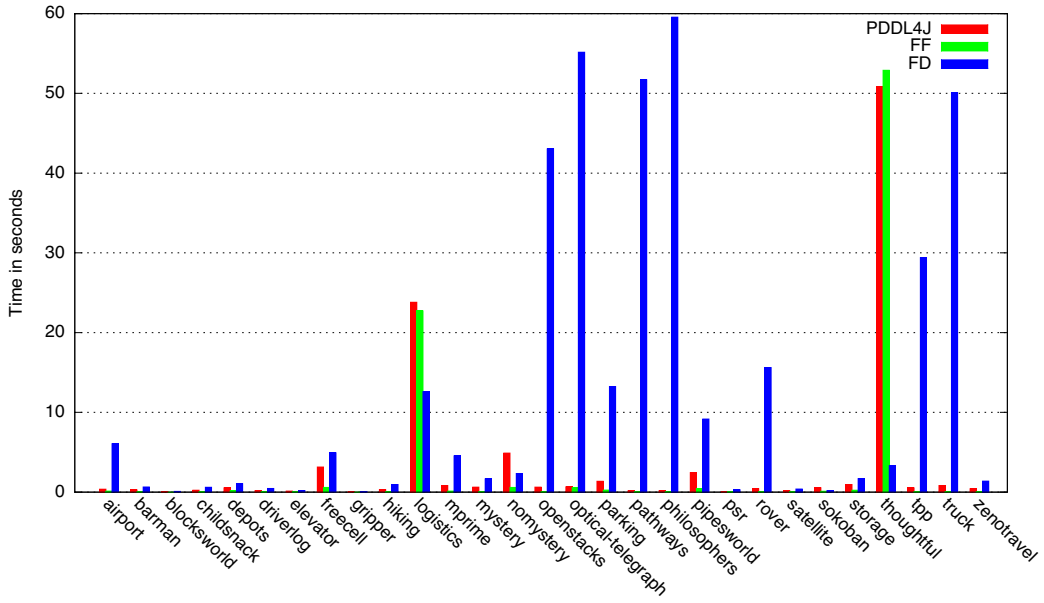


FIGURE 10.10 – Temps moyen pour instancier et encoder les problèmes de planification

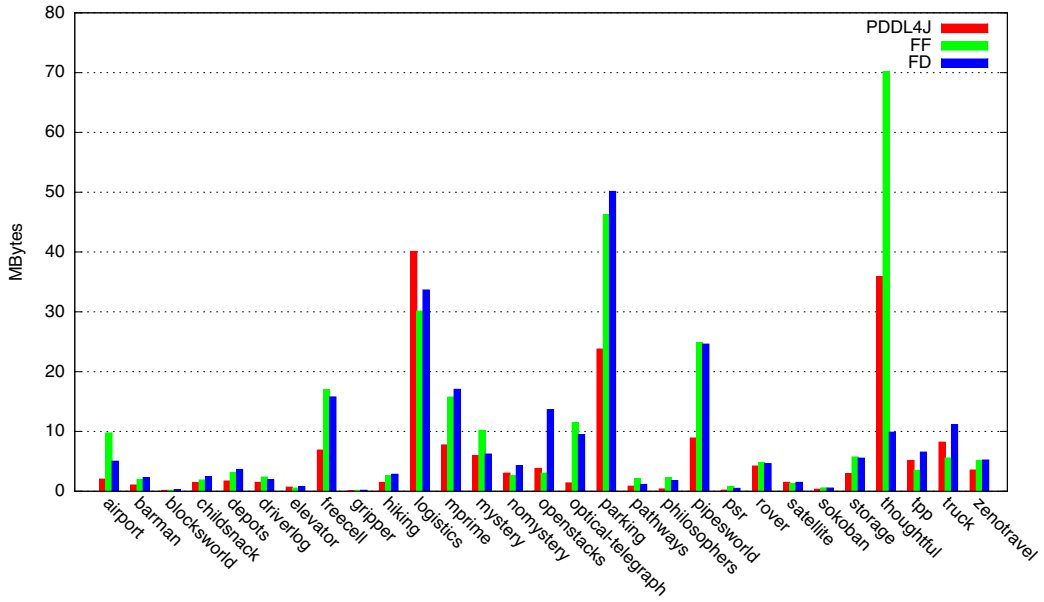


FIGURE 10.11 – Taille mémoire moyenne nécessaire au stockage des problèmes de planification

1. L'ajout de nouveaux planificateurs à l'état de l'art et l'implémentation de nouvelles heuristiques afin d'intégrer les algorithmes et techniques les plus performants et compétitifs à PDDL4J.
2. Le développement des interfaces de PDDL4J pour faciliter son intégration en particulier avec :
   (a) les solveurs SAT et CSP tels que SAT4J et Choco que nous avons déjà évoqués afin de renforcer les liens vers ces communautés connexes et

(b) ROS (*Robot Operating System*) (QUIGLEY, GERKEY et SMART, 2015) pour permettre l'utilisation aisée de PDDL4J dans des architectures robotiques. En effet, ROS est un framework *open source* incontournable en robotique qui a pour objectif d'offrir une couche d'abstraction au-dessus des architectures robotiques matérielles. ROS propose également une collection d'outils pour la programmation de robots et est utilisé par des centaines d'équipes de recherche. Une version bêta du module ROS que nous avons développée et qui intègre PDDL4J est disponible en *open source* [6].

6. https://github.com/pellierd/pddl4j_rospy

Chapitre 11

Conclusion et perspectives

Ce mémoire a présenté nos recherches portant sur la planification et la conduite de systèmes autonomes. L'objectif de nos recherches est de développer des méthodes et des techniques génériques et indépendantes du domaine afin de concevoir des systèmes autonomes capables de décider quelles actions réaliser et comment les exécuter pour atteindre un objectif défini *a priori*.

11.1 Synthèse de mes contributions

Nous avons présenté une lecture de nos travaux sous quatre angles différents. Dans un premier temps, nous avons présenté plusieurs contributions en considérant le problème de la conduite des systèmes autonomes comme un problème de planification en boucle fermée, i.e., lorsque planification et exécution sont entrelacées. Nous avons traité cette problématique spécifique en proposant plusieurs techniques pour planifier en temps réel. Nous avons ainsi considéré le cas où la planification est un composant parmi d'autres au sein d'une architecture complexe et où le temps alloué à la prise de décision est limité (Pellier, Bouzy et Métivier, 2010a,b, 2011). Nous avons également présenté une approche originale pour prendre en compte le fait que le but peut évoluer pendant la recherche d'un plan solution ou encore pendant son exécution (Pellier, Fiorino et Métivier, 2013, 2014). Ces techniques ont été utilisées notamment dans l'architecture robotique CAIO que nous avons développée au sein de l'équipe Magma du Laboratoire d'Informatique de Grenoble financé par l'ANR MoCa (*Cognitive and Affective Architecture for Social Human-Robot Interaction*) (Adam et al., 2016 ; W.Johal et al., 2015).

Dans un second temps, nous avons présenté trois contributions à la planification distribuée. Dans ces travaux, nous nous sommes intéressés à la distribution du processus de planification sur plusieurs agents pour atteindre un objectif commun. Nous avons fait l'hypothèse qu'aucun des agents impliqués dans le processus de planification n'était capable d'atteindre l'objectif seul. Par conséquent, les agents sont dans l'obligation de coopérer en partageant leurs compétences et leurs connaissances. La première contribution repose sur une recherche distribuée dans des graphes de planification (Pellier, 2010). Nous avons montré que cette approche permettait de résoudre des problèmes de planification où les actions des différents agents sont fortement entrelacées. Nous avons également proposé une approche de planification distribuée tirant parti de deux techniques de planification centralisée (Pellier et Fiorino, 2005b, 2007, 2009) : POP (*Partial Order Planning*) qui se prête bien à la distribution du processus de planification, car elles ne nécessitent pas la représentation explicite d'état, et la planification hiérarchique aussi appelée HTN (*Hierachical Task Networks*) pour son expressivité et ses algorithmes performants. Finalement, nous avons proposé une approche distribuée domaine spécifique pour

résoudre le problème de poursuite-évasion (PELLIER et FIORINO, 2005a) permettant d'optimiser le nombre d'agents nécessaires à la surveillance d'environnements labyrinthiques.

Dans un troisième temps, nous avons proposé deux contributions à la planification en termes d'heuristiques. Nous avons montré notamment comment il était possible d'apprendre de manière incrémentale des macro-actions, i.e., des séquences d'actions fréquemment utilisées, à partir des plans solution, et comment les utiliser pour accélérer la recherche de nouveaux plans (CASTELLANOS-PAEZ et al., 2016; DULAC et al., 2013). Nous avons également proposé une méthode heuristique pour l'encodage et la simplification des problèmes de planification HTN (RAMOUL et al., 2016, 2017).

Finalement dans la dernière partie de ce manuscrit, nous avons présenté une contribution technique à la planification automatique en décrivant la librairie PDDL4J [1] (PELLIER et FIORINO, 2017). L'objectif de cette librairie est double : d'une part, permettre à la communauté de développer plus facilement de nouvelles techniques de planification et d'autre part permettre une diffusion plus large des techniques de planification et en favoriser les débouchés industriels.

11.2 Perspectives

Tout au long de ce manuscrit, nous avons évoqué des pistes pour étendre les différents travaux présentés. Nous souhaitons dans cette conclusion aller plus loin en donnant les grandes lignes de nos futurs travaux ainsi que le contexte applicatif qui nous permettra de les tester et de les valider. Nous défendons l'idée qu'il faut dépasser la problématique de l'amélioration des performances, qui a toujours été au cœur des recherches en planification. Bien évidemment, l'amélioration reste la clé si l'on veut résoudre des problèmes de plus en plus complexes. Toutefois, le domaine est dorénavant suffisamment mature pour réfléchir en termes d'interfaces et d'usages. Selon nous, la planification est confrontée à deux défis majeurs : l'apprentissage de modèles d'action et l'intégration de l'humain dans la boucle de planification. Nous avons fait le choix d'inscrire dans notre projet de recherche ces deux défis dans un contexte applicatif commun : la programmation de robots collaboratifs (cobots) en milieux industriels.

11.2.1 L'apprentissage de modèles d'actions

Planifier n'est possible que si le problème à résoudre est correctement modélisé en termes d'actions, d'état initial et de buts. Le développement d'un langage comme PDDL pour représenter les connaissances nécessaires à la modélisation d'un problème de planification a été une étape importante pour disséminer et populariser les techniques de planification. Toutefois, à ce jour, malgré les efforts réalisés (SHAH et al., 2013), il existe peu d'outils et de méthodes pour aider à l'acquisition des connaissances nécessaires à la modélisation d'un problème de planification. La modélisation reste très largement effectuée manuellement par un expert humain et est par conséquent extrêmement coûteuse. De plus, s'il est possible de réaliser cette modélisation pour des problèmes relativement simples, ce processus reste complexe lorsqu'il s'agit de modéliser des problèmes réels. Le développement de techniques automatiques ou semi-automatiques d'apprentissage de modèles d'actions, pouvant

1. http://pddl4j.imag.fr

être utilisés en entrée d'un planificateur à partir d'observations ou de traces, est donc un verrou majeur freinant la dissémination des techniques de planification.

La littérature sur le sujet (ARORA et al., 2018 ; ZHUO et YANG, 2014) montre que peu d'algorithmes d'apprentissage sont capables d'apprendre un modèle d'actions suffisamment complet pour être réellement utilisable sans l'intervention d'un expert humain. Par ailleurs, les modèles d'actions appris restent relativement simples. Ils reposent pour la plupart sur le langage STRIPS. Par conséquent, il reste de nombreuses recherches à mener pour à la fois améliorer la qualité des modèles appris et enrichir leur expressivité, en intégrant par exemple le temps ou encore la concurrence au sein des modèles. Ces deux aspects n'ont pas du tout été abordés dans la littérature. Nous avons récemment proposé, dans le cadre d'un co-encadrement de thèse financé par l'ANR Sombrero, une contribution préliminaire dans ce sens (ARORA et al., 2017).

11.2.2 Planifier en initiative mixte

Parler de systèmes intelligents et autonomes n'implique pas que les systèmes soient déconnectés de l'humain. Bien au contraire, la grande majorité des systèmes intelligents développés le sont pour aider l'homme dans la réalisation d'une tâche. Cela sous-entend qu'ils sont en forte interaction avec lui. Face à ce constat, de nouveaux droits apparaissent comme le *droit à l'explication* pour les décisions algorithmiques. Ce nouveau droit est mis en avant par la loi pour une République Numérique, ainsi que par le règlement général sur la protection des données GDPR (*General Data Protection Regulation*), qui entrent en vigueur pour les membres de l'Union Européenne en mai 2018. Même si des questions juridiques restent ouvertes quant à la portée et aux conséquences de ces textes, cette notion constitue un défi important, en particulier pour les systèmes intelligents[2]. La planification, en tant que raisonnement sur l'action, peut donc jouer un rôle important dans ce contexte.

Nous défendons l'idée qu'une des clés pour résoudre ce problème réside dans la capacité des systèmes planifiants à co-construire avec l'humain à la fois une représentation commune des actions élémentaires à réaliser, mais également un plan solution partagé. C'est au travers de cette co-construction que les décisions prises par les systèmes autonomes peuvent être rendues intelligibles et acceptables pour et par leurs utilisateurs. Le développement de systèmes planifiants en initiative mixte doit permettre de tirer parti des forces de la machine à explorer une multitude de possibilités, et de l'humain et de sa capacité de synthèse. Un certain nombre de systèmes reprenant cette idée ont été développés, e.g., (BRESINA et MORRIS, 2007 ; MYERS et al., 2002), etc. Toutefois, aucun de ces systèmes n'intègre les deux aspects liés à co-construction précédemment cités et aucun n'est capable de justifier ses décisions et encore moins les raisons de ses échecs. Il reste donc beaucoup de recherches à mener pour adapter des algorithmes de planification existants ou en développer de nouveaux dotés de telles capacités. Il est important de souligner que ces travaux intéressent d'ores et déjà les industriels. À titre d'exemple, nous avons récemment entrepris une collaboration avec la société Cloud Temple en charge de la gestion d'infrastructures de *cloud computing*. L'objectif de cette collaboration est de développer un outil d'aide à la décision assistant les administrateurs dans leurs tâches quotidiennes de gestion. Pour cette société, le développement d'un outil d'aide à la décision reposant sur des algorithmes de planification en initiative mixe revêt un caractère extrêmement stratégique. L'outil doit lui permettre de faire

2. https://en.wikipedia.org/wiki/Right_to_explanation

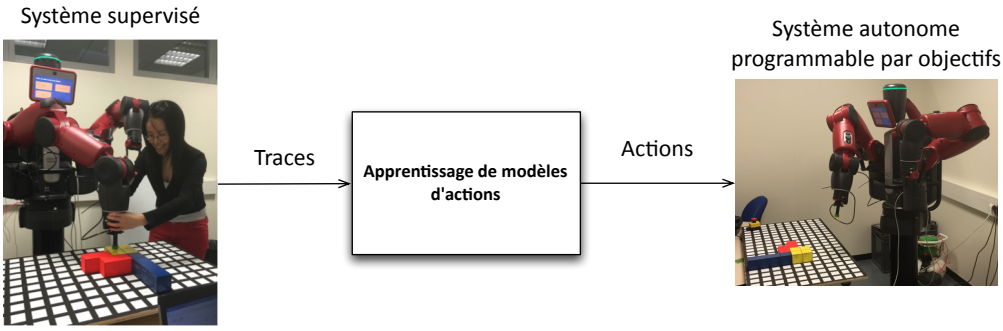


FIGURE 11.1 – À gauche, un opérateur utilise un robot (le robot Baxter) pour réaliser une tâche. Le robot n'a aucune autonomie. Il effectue la tâche guidée par l'opérateur. Les traces représentent les changements d'état du monde observés par le robot. Au centre de la figure, le modèle d'actions est inféré à partir des traces d'exécution produites par l'opérateur. À droite, le système décide et agit de manière autonome par planification comment atteindre l'objectif partagé défini par l'opérateur à partir du modèle d'actions appris.

face à l'accroissement important de la taille des infrastructures matérielles rendant leur administration de plus en plus complexe.

11.2.3 Cas d'étude : la programmation de cobots en milieu industriel

Pour tester et valider nos travaux, nous avons fait le choix de les inscrire dans le contexte applicatif de la programmation de cobots en milieu industriel. En effet, la robotique a profondément bouleversé le monde de l'industrie. Dans certains secteurs (automobile, pharmaceutique, etc.), les robots industriels ont complètement remplacé les ouvriers sur les chaînes de production à des tâches élémentaires, répétitives, mécaniques et à forte pénibilité (par exemple, le moulage, le sablage ou l'assemblage de pièces dont le processus est déterministe et sans aléas). La cobotique, ou robotique collaborative en milieu industriel, a pour objectif de faire *sortir le robot industriel de sa cage de protection* et de le faire travailler avec des opérateurs humains sur un même poste de travail pour la réalisation de tâches complexes combinant efforts physiques et expertises humaines (par exemple le dévracage, le conditionnement et la palettisation de pièces qui nécessitent la prise en compte du contexte et de ses aléas). La cobotique est donc une nouvelle étape très importante pour le développement de l'usine du futur, car elle permettra d'associer les capacités cognitives de l'Homme avec la productivité de la Machine. Par exemple, les robots collaboratifs ou cobots, par leurs capacités polyvalentes et reprogrammables, auront vocation à répondre au besoin d'adaptabilité des chaînes de production des petites et moyennes industries (PMI), soumises au renouvellement de plus en plus rapide des gammes de produits et à une demande croissante de produits de plus en plus personnalisés. Toutefois, le principal verrou est celui de la programmation des cobots : actuellement, la programmation des robots nécessite l'intervention lourde et coûteuse d'ingénieurs-roboticiens très spécialisés et externes à la PMI, les intégrateurs robotiques. Ce qui est contradictoire avec les besoins d'adaptabilité des PMI et une cobotique au plus près des chaînes de production.

On retrouve dans ce contexte applicatif (cf. figure 11.1) les deux défis précédemment mentionnés. Il faut à la fois être capable d'apprendre aux cobots les actions élémentaires à réaliser, autrement dit lui apprendre un modèle d'actions, et d'autre part être capable de planifier avec lui l'ensemble des actions à réaliser

pour effectuer une tâche partagée complexe. Nous avons obtenu très récemment des résultats nous encourageant dans cette voie[3] montrant la pertinence de nos approches et de nos contributions (LIANG et al., 2017a,b).

3. https://www.youtube.com/watch?v=XOeR0QnYgM4

Bibliographie

ADAM, C. et al. (2016). « Social Human-Robot Interaction : A New Cognitive and Affective Interaction-Oriented Architecture ». In : *Proceedinsg of the International Conference on Social Robotics*, p. 253–263.

ALFORD, R., P. BERCHER et D. AHA (2015). « Tight Bounds for HTN Planning with Task Insertion ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1502–1508.

ARORA, A. et al. (2017). « Action Model Acquisition Using Sequential Pattern Mining ». In : *Proceedings of the German Conference on AI*, p. 286–292.

ARORA, A. et al. (2018). « A Review on Learning Planning Action Models ». In : *The Knowledge Engineering Review*.

ASAI, M. et A. FUKUNAGA (2015). « Solving Large-Scale Planning Problems by Decomposition and Macro Generation ». In : *Proceedings of International Conference on Automated Planning and Scheduling*, p. 16–24.

ASUNCIÓN, M. de la et al. (2005). « SIADEX : An interactive knowledge-based planner for decision support in forest fire fighting ». In : *Artificial Intelligence Communications* 18.4, p. 257–268.

AUER, P., N. CESA-BIANCHI et P. FISHER (2002). « Finite-time Analysis of the Multiarmed Bandit Problem ». In : *Machine Learning* 47.2–3, p. 235–256.

AUSIELLO, G. et al. (1999). *Complexity and Approximation*. Springer-Verlag.

BAR-NOY, A. et al. (2001). « A unified approach to approximating resource allocation and scheduling ». In : *Journal of Association for Computing Machinery* 48.5, p. 1069–1090.

BARTÁK, R., M. SALIDO et F. ROSSI (2010). « Constraint satisfaction techniques in planning and scheduling ». In : *Journal of Intelligent Manufacturing* 21.1, p. 5–15.

BARTÁK, R. et D. TOROPILA (2008). « Reformulating constraint models for classical planning ». In : *Proceedings of The International Florida Artificial Intelligence Research Society Conference*, p. 525–530.

BENDA, M., V. JAGANNATHAN et R. DODHIAWALA (1986). *On optimal cooperation of knowledge sources - an empirical investigation*. Rapp. tech. BCS-G2010-28. Boeing Advanced Technology Center, Boeing Computing Services.

BERNARDINI, S., M. FOX et D. LONG (2014). « Planning the Behaviour of Low-Cost Quadcopters for Surveillance Missions ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 445–453.

BEVACQUA, G et al. (2015). « Mixed-Initiative Planning and Execution for Multiple Drones in Search and Rescue Missions. » In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 315–323.

BEVILACQUA, L. et al. (2011). « A tool for automatic generation of WS-BPEL compositions from OWL-S described services ». In : *Proceedings of the International Conference on Software, Knowledge Information, Industrial Management and Applications*.

BHADAURIA, D. et V. ISLER (2011). « Capturing an Evader in a Polygonal Environment with Obstacles ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 2054–2059.

BJARNASON, R., A. FERN et P. TADEPALLI (2009). « Lower Bounding Klondike Solitaire with Monte-Carlo Planning ». In : *Proceedings of the International Conference on Planning and Scheduling*, p. 26–33.

BLUM, A. et M. FURST (1997). « Fast Planning Through Planning Graph Analysis ». In : *Artificial Intelligence* 90.1-2, p. 281–300.

BONET, B. et H. GEFFNER (1999). « Planning as Heuristic Search : New Results ». In : *Proceedings of the European Conference on Planning*, p. 359–371.

— (2001). « Planning as heuristic search ». In : *Artificial Intelligence* 129.1–2, p. 5–33.

BORIE, R., C. TOVEY et S. KOENIG (2011). « Algorithms and complexity results for graph-based pursuit evasion ». In : *Autonous Robots* 31.4, p. 317–332.

BOTEA, A., M. MÜLLER et J. SCHAEFFER (2005). « Learning Partial-Order Macros from Solutions ». In : *Proceedings of the International Conference on Planning and Scheduling*, p. 231–240.

— (2007). « Fast Planning with iterative Macros ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1828–1833.

BOTEA, A. et al. (2005). « Macro-FF : Improving AI Planning with Automatically Learned Macro-Operators ». In : *Journal of Artificial Intelligence Research* 24, p. 581–621.

BRAFMAN, R. et C. DOMSHLAK (2003). « Structure and Complexity in Planning with Unary Operators ». In : *Journal of Artificial Intelligence Research* 18.1, p. 315–349.

— (2006). « Factored Planning : How, When, and When Not ». In : *Proceedings of the Association for the Advancement of Artificial Intelligence*, p. 809–814.

— (2008). « From One to Many : Planning for Loosely Coupled Multi-Agent Systems ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 28–35.

BRENNER, M. (2003). « A Multiagent Planning Language ». In : *Proceedings of the ICAPS Workshop on Planning Domain Description Language*.

BRESINA, J. et P. MORRIS (2007). « Mixed-initiative planning in space mission operations ». In : *AI magazine* 28.2, p. 75.

BRYCE, D. et S. KAMBHAMPATI (2007). « A Tutorial on Planning Graph Based Reachability Heuristics ». In : *Artificial Intelligence Magazine* 27.1, p. 47–83.

BULITKO, V. et G. LEE (2006). « Learning in Real-Time Search : A Unifying Framework ». In : *Journal of Artificial Intelligence Research* 25, p. 119–157.

BULITKO, Vadim (2004). *Learning for Adaptive Real-Time Search*. Rapp. tech. Department of Computer Science, University of Alberta.

BUTLER, R. et C. MUÑOZ (2006). *An Abstract Plan Preparation Language*. Rapp. tech. TM-2006-214518. NASA.

CARDOSO, R. et R. BORDINI (2016). « A Distributed Online Multi-Agent Planning System ». In : *Proceedings of the Workshop on Distributed and Mutiltiagent Planning (ICAPS)*, p. 15–23.

CARLOS, L L., J C. SERGIO et H. MALTE (2013). « Automating the evaluation of planning systems ». In : *Artificial Intelligence Commun.* 26.4, p. 331–354.

CASTELLANOS-PAEZ, S. et al. (2016). « Learning Macro-actions for State-Space Planning ». In : *Proceedings of the International Conference on Artificial Intelligence and Pattern Recognition*.

CHASLOT, G. et al. (2006). « Monte-Carlo Tree Search in Production Management Problems ». In : *Proceedings of the BeNeLux Conference on Artificial Intelligence*, p. 91–98.

CHASLOT, G. et al. (2008). « Progressive Strategies for Monte-Carlo Tree Search ». In : *New Mathematics and Natural Computation* 4.3, p. 343–357.

CHEN, H. et O. GIMÉNEZ (2010). « Causal graphs and structurally restricted planning ». In : *Journal of Computer System and Science* 10, p. 273–314.

CHENG, C-H. et al. (2011). « GAVS+ : An Open Platform for the Research of Algorithmic Game Solving ». In : *Proceedings of the Joint European Conferences on Theory and Practice of Software*, p. 258–261.

CHRPA, L., M. VALLATI et T. MCCLUSKEY (2014). « MUM : A Technique for Maximising the Utility of Macro-operators by Constrained Generation and Use ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*.

— (2015). « On the Online Generation of Effective Macro-Operators ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1544–1550.

CHUNG, T., G. HOLLINGER et V. ISLER (2011). « Search and pursuit-evasion in mobile robotics - A survey ». In : *Autonous Robots* 31.4, p. 299–316.

CLEMENT, B. et A. BARRETT (2003). « Continual Coordination through Shared Activities ». In : *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, p. 57–67.

COLES, A. et A. SMITH (2007). « Marvin : A Heuristic Search Planner with Online Macro-Action Learning ». In : *Journal of Artificial Intelligence Research* 28.1.

COOPER, M., M. de ROQUEMAUREL et P. RÉGNIER (2011). « A weighted CSP approach to cost-optimal planning ». In : *AI communication* 24.1, p. 1–29.

COX, J. et E. DURFEE (2005). « An efficient algorithm for multiagent plan coordination ». In : *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, p. 828–835.

COX, M. et M. VELOSO (1998). « Goal Transformations in Continuous Planning ». In : *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning*.

CRASS, D., I. SUZUKI et M. YAMASHITA (1995). « Searching for a mobile intruder in a corridor - the open edge variant of the polygon search problem ». In : *International Journal of Computational Geometry and Applications* 5(4), p. 397–412.

CURRIE, K. et A. TATE (1991). « O-Plan : the open planning architecture ». In : *Artifical Intelligence* 52.1, p. 49–86.

DECHTER, R. (2003). *Constraint Processing*. Morgan Kaufmann.

DIGANI, V. et al. (2014). « An automatic approach for the generation of the roadmap for multi-AGV systems in an industrial environment ». In : *2IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 1736–1741.

D'INVERNO, M. et al. (2004). « The dMARS Architecture : A Specification of the Distributed Multi-Agent Reasoning System ». In : *Autonomous Agents and Multi-Agent Systems* 9.1-2, p. 5–53.

DOMSHLAK, C., J. HOFFMANN et M. KATZ (2015). « Red-black planning : A new systematic approach to partial delete relaxation ». In : *Artificial Intelligence* 221, p. 73–114.

DULAC, A. et al. (2013). « Learning Useful Macro-actions for Planning with N-Grams ». In : *Proceedinsg of the International Conference on Tools with Artificial Intelligence*, p. 803–810.

DURKOTA, K. et A. KOMENDA (2013). « Deterministic Multiagent Planning Techniques : Experimental Comparison ». In : *Proceedings of the Distributed and Multi-agent Planning Workshop part of the ICAPS Conference*.

EDELKAMP, S. (2001). « Planning with Pattern Databases ». In : *Proceedings of the European conference on Planning*, p. 13–24.

EDELKAMP, S. et J. HOFFMANN (2004). *PDDL2.2 : The Language for the Classical Part of the 4th International planning Competition*. Rapp. tech. 195. Institut für Informatik.

EROL, K., J. HENDLER et D. NAU (1994). « HTN Planning : Complexity and Expressivity ». In : *Proceedings of the National Conference on Artificial Intelligence*, p. 1123–1128.

FIKES, R., P. HART et N. NILSSON (1972). « Learning and executing generalized robot plans ». In : *Artificial Intelligence* 3.4, p. 251–288.

FIKES, R. et N. NILSSON (1971). « STRIPS : A new approach to the application of theorem proving to problem solving ». In : *Artificial Intelligence* 3-4.2, p. 189–208.

FISHER, F. et al. (2000). « A planning approach to monitor and control for deep space communications ». In : *Proceedings of the Confrence on Aerospace*, p. 311–320.

FOURNIER-VIGER, P. et al. (2017). « A Survey of Sequential Pattern Mining ». In : *Journal of Data Science and Pattern Recognition* 1.1, p. 54–77.

FOX, M. et D. LONG (2002). « PDDL+ : Modelling Continuous Time-dependent Effects ». In : *Proceedings of the International NASA Workshop on Planning and Scheduling*.

— (2003). « PDDL 2.1 : An Extension to PDDL for Expressing Temporal Planning Domains ». In : *Journal of Artificial Intelligence Research* 20.1, p. 61–124.

— (2006). « Modelling Mixed Discrete-Continuous Domains for Planning ». In : *Journal of Artificial Intelligence Research* 27, p. 235–297.

FOX, M. et al. (2006). « Plan Stability : Replanning versus Plan Repair ». In : *Proceedings of the International Conference on Planning and Scheduling*, p. 212–221.

FRANK, J. et A. JÓNSSON (2003). « Constraint-Based Attribute and Interval Planning ». In : *Constraints* 8.4, p. 339–364.

FURCY, D. et S. KOENIG (2000). « Speeding up the Convergence of Real-Time Search ». In : *Proceedings of the Association for the Advancement of Artificial Intelligence*, p. 891–897.

GAUTHAM, D. et al. (2015). « A Distributed Task Allocation Algorithm for a Multi-Robot System in Healthcare Facilities ». In : *Journal of Intelligent & Robotic Systems* 80.1, p. 33–58.

GELLY, S. et al. (2006). *Modification of UCT with Patterns in Monte-Carlo Go*. Rapp. tech. RR-6062. INRIA.

GEORGIEVSKI, I. et A. MARCO (2015). « HTN planning : Overview, comparison, and beyond ». In : *Artificial Intelligence* 222, p. 124–156.

GEREVINI, A. et D. LONG (2005a). « BNF Description of PDDL3.0 ». Unpublished manuscript from the International Planning Competition website.

— (2005b). *Plan Constraints and Preferences in PDDL3*. Rapp. tech. 2005-08-47. Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia.

— (2005c). « Preferences and Soft Constraints in PDDL3 ». In : *Proceedings of the ICAPS Workshop on Preferences and Soft Constraints in Planning*, p. 46–54.

GEREVINI, A. et L. SCHUBERT (1996). « Accelerating Partial-Order Planners : Some Techniques for Effective Search Control and Pruning ». In : *Journal of Artificial Intelligence Research* 5.1, p. 95–137.

GHALLAB, M., D. NAU et P. TRAVERSO (2004). *Automated Planning Theory and Practice*. Sous la dir. d'ELSEVIER. Morgan Kaufmann Publishers.

GHALLAB, M. et al. (1998). *PDDL : The Planning Domain Definition Language*. International Planning Competition Commitee.

GUIBAS, L. et al. (1999). « A visibility-based pursuit-evasion problem ». In : *International Journal of Computational Geometry and Applications* 9, p. 471–493.

HAN, J., M. KAMBER et J. PEI (2011). *Data Mining : Concepts and Techniques*. Morgan Kaufman.

HART, P., N. NILSSON et B. RAPHAEL (1968). « A formal basis for the heuristic determination of minimum cost paths ». In : *IEEE Transactions on Systems, Man and Cybernetics* 2, p. 100–107.

HASLUM, P., B. BONET et H. GEFFNER (2005). « New Admissible Heuristics for Domain-Independent Planning ». In : *Proceedings of the Association for the Advancement of Artificial Intelligence*, p. 1163–1168.

HASLUM, P. et H. GEFFNER (2000). « Admissible Heuristics for Optimal Planning ». In : *Proceedings of the Artificial Intelligence Planning Systems*, p. 140–149.

HASLUM, P. et al. (2007). « Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning ». In : *Proceedings of the Association for the Advancement of Artificial Intelligence*, p. 1007–1012.

HEAPS, H. (1978). *Information retrieval : computational and theoretical aspects*. Academic Press.

HELMERT, M. (2006). « The Fast Downward Planning System ». In : *Journal of Artificial Intelligence Research* 26, p. 191–246.

— (2008). « Changes in PDDL 3.1 ». Unpublished manuscript International Planning Competition website.

HELMERT, M. et C. DOMSHLAK (2009). « Landmarks, Critical Paths and Abstractions : What is the difference anyway ». In : *Proceedings of the International Conference on Planning and Scheduling*, p. 162–171.

HELMERT, M. et al. (2014). « Merge-and-Shrink Abstraction : A Method for Generating Lower Bounds in Factored State Spaces ». In : *Journal of the ACM* 61.3, p. 16–63.

HERNÁNDEZ, C., S. CONCEPCIÓN et P. MESEGUER (2009). « Lookahead, Propagation and Moves in Real-Time Heuristic Search ». In : *Proceedings International Symposium on Combinatorial Search*.

HOFFMANN, J. (2011). « Analyzing search topology without running any search : On the connection between causal graphs and h+ ». In : *Journal of Artificial Intelligence Research* 41, p. 155–229.

HOFFMANN, J. et B. NEBEL (2001). « The FF Planning System : Fast Plan Generation Through Heuristic Search ». In : *Journal of Artificial Intelligence Research* 14.1, p. 253–302.

HOFFMANN, J., J. PORTEOUS et L. SEBASTIA (2004). « Ordered Landmarks in Planning ». In : *Journal of Artifcial Intelligence Research* 22, p. 215–278.

HOLLINGER, G., A. KEHAGIAS et S. SINGH (2007). « Probabilistic Strategies for Pursuit in Cluttered Environments with Multiple Robots ». In : *Proceedings of the International Conference on Robotics and Automation*, p. 3870–3876.

HOLLINGER, G., S. SINGH et A. KEHAGIAS (2010). « Improving the Efficiency of Clearing with Multi-agent Teams ». In : *International Journal of Robotics Research* 29.8, p. 1088–1105.

HOLLINGER, G. et al. (2009). « Efficient Multi-robot Search for a Moving Target ». In : *International Journal of Robotics Research* 28.2, p. 201–219.

HOWEY, R., D. LONG et M. FOX (2004). « VAL : Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL ». In : *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, p. 294–301.

HSU, C-W., B. WAH et Y. CHEN (2005). « Subgoal Ordering and Granularity Control for Incremental Planning ». In : *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, p. 507–514.

IBA, G. (1989). « A heuristic approach to the discovery of macro-operators ». In : *Machine Learning* 3.4, p. 285–317.

ISHIDA, T. (1998). « Real-time search for autonomous agents and multiagent system ». In : *Autonomous Agents and Multi-Agent Systems* 1.2, p. 139–167.

ISHIDA, T. et R. KORF (1991). « Moving target search ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 204–210.

— (1995). « A realtime search for changing goals ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6.17, p. 609–619.

IWEN, M. et A. D. MALI (2002). « Automatic Problem Decomposition for Distributed Planning ». In : *Proceedings of the International Conference on Artificial Intelligence*, p. 411–417.

JAGANNATHAN, V., R. DODHIAWALA et L. BAUM (1989). *Blackboard Architectures and Applications*. Academic Press.

JONSSON, A. (2007). « The Role of Macros in Tractable Planning Over Causal Graphs ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1936–1941.

— (2009). « The Role of Macros in Tractable Planning ». In : *Journal of Artificial Intelligence Research* 36, p. 471–511.

JONSSON, P. et C. BÄCKSTRÖM (1998). « Tractable plan existence does not imply tractable plan generation ». In : *Annals of Mathematics and Artificial Intelligence* 22.3–4, p. 281–296.

KAMBHAMPATI, S. (2000). « Planning Graph as a (Dynamic) CSP : Exploiting EBL, DDB and other CSP Search Techniques in Graphplan ». In : *Journal of Artificial Intelligence Research* 12.1, p. 1–34.

KAMBHAMPATI, S. et J. A. HENDLER (1992). « A validation-structure-based theory of plan modification and reuse ». In : *Artifificial Intelligence* 55, p. 193–258.

KATZ, M. et C. DOMSHLAK (2008). « Structural Patterns Heuristics via Fork Decomposition ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 182–189.

KAUTZ, H. et B. SELMAN (1992). « Planning as Satisfiability ». In : *Proceedings of the European Conference on Artificial Intelligence*, p. 359–363.

— (1999). « Unifying SAT-based and Graph-based Planning ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 318–325.

KAUTZ, H., B. SELMAN et J. HOFFMANN (2006). « SatPlan : Planning as Satisfiability ». In : *Abstracts of the 5th International Planning Competition*.

KEHAGIAS, A., G. HOLLINGER et S. SINGH (2009). « A graph search algorithm for indoor pursuit/evasion ». In : *Mathematical and Computer Modelling* 50.9-10, p. 1305–1317.

KICHKAYLO, T. et al. (2007). « Mixed-Initiative Planning for Space Exploration Missions ». In : *Proceedings of the International Workshop on Moving Planning and Scheduling Systems into the Real World (ICAPS)*.

KNOBLOCK, C. (1994). « Automatically generating abstractions for planning ». In : *Artificial intelligence* 68.2, p. 243–301.

KOCSIS, L. et C. SZEPESVARI (2006). « Bandit-based Monte-Carlo Planning ». In : *Proceedings of the European Conference on Machine Learning*, p. 282–293.

KOEHLER, J. et J. HOFFMANN (1999). *Handling of inertia in a planning system*. Rapp. tech. Albert-Ludwigs University at Freiburg.

— (2000). « On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm ». In : *Journal of Artifcial Intelligence Research* 12, p. 339–386.

KOEHLER, J. et al. (1997). « Extending planning graphs to an ADL subset ». In : *Proceedings of the European Conference on Planning*, p. 273–285.

KOENIG, S. et M. LIKHACHEV (2002). « D* Lite ». In : *Proceedings of the Association for the Advancement of Artificial Intelligence*, p. 476–483.

— (2005). « Adaptive A* ». In : *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, p. 1311–1312.

KOENIG, S., M. LIKHACHEV et X. SUN (2007). « Speeding up moving target search ». In : *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, p. 188–197.

KOENIG, S. et X. SUN (2009). « Comparing Real-Time and Incremental Heuristic Search for Real-Time Situated Agents ». In : *Journal of Autonomous and Multi-agent System* 18.3, p. 313–341.

KORF, R. (1985). « Macro-operators : A weak method for learning ». In : *Artificial Intelligence* 26.1, p. 35–77.

— (1990). « Real-Time Heuristic Search ». In : *Artificial Intelligence* 42.2-3, p. 189–211.

KOVACS, L. (2012). « A Multi-Agent Extension of PDDL3.1 ». In : *Proceedings of the Workshop on the International Planning Competition*, p. 19–27.

KROGT, R. van der et M. de WEERDT (2005). « Plan Repair as an Extension of Planning ». In : *Proceedings of the International Conference on Planning and Scheduling*, p. 161–170.

KUTER, U. et al. (2009). « Task decomposition on abstract states, for planning under nondeterminism ». In : *Artificial Intelligence* 173.5-6, p. 669–695.

LALLEMENT, R., L. DE SILVA et R. ALAMI (2014). « HATP : An HTN planner for robotics ». In : *Proceedings of the International Workshop on Planning and Robotics (ICAPS)*.

LAU, H., Huang S. et G. DISSANAYAKE (2005). « Optimal search for multiple targets in a built environment ». In : *IEEE/RSJ International Conference on Intelligent Robots and Systems*, p. 3740–3745.

LAVALLE, S. et J. HINRICHSEN (2001). « Visibility-based pursuit-evasion : the case of curved environments ». In : *IEEE Trans. Robotics and Automation* 17.2, p. 196–202.

LAZEBNIK, S. (2001). *Visibility-based pursuit-evasion in three-dimentional environments*. Rapp. tech. CVR-TR-2001-01.

LE BERRE, D. et A. PARRAIN (2010). « The SAT4J library, release 2.2 ». In : *Journal on Satisfiability, Boolean Modeling and Computation* 7.59–64.

LESSER, V. et al. (2004). « Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework ». In : *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, p. 87–143.

LEVY, R. et J. ROSENSCHEIN (1992). « A game theoretic approach to distributed artificial intelligence and the pursuit problem ». In : *SIGOIS Bull.* 13.3, p. 11.

LIANG, R. (2012). « A Survey of Heuristics for Domain-Independent Planning ». In : *Journal of Software* 7.9, p. 2099–2016.

LIANG, Y-S. et al. (2017a). « A Framework for Robot Programming in Cobotic Environments : First user experiments ». In : *Proceedings of the International Conference on Mechatronics and Robotics Engineering*, p. 30–35.

— (2017b). « Evaluation of a Robot Programming Framework for Non-Experts Using Symbolic Planning Representations ». In : *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication*, p. 1121–1126.

LIU, D. et T. L. MCCLUSKEY (2001). *The OCL Language Manual*. Department of Computing et Mathematical Sciences, University of Huddersfield.

LONG, D. et M. FOX (1999). « Efficient Implementation of the plan graph in STAN ». In : *Journal of Artificial Intelligence Research* 10.87–115.

LOPEZ, A. et F. BACCHUS (2003). « Generalizing GraphPlan by Formulating Planning as a CSP ». In : *Proceedings of the International Conference on Artificial Intelligence*, p. 954–960.

LUIS, N. et D. BORRAJO (2014). « Plan Merging by Reuse for Multi-Agent Planning ». In : *Proceedings of the Workshop on Distributed and Mutiltiagent Planning (ICAPS)*, p. 38–46.

MARECKI, J. et M. TAMBE (2008). « Towards Faster Planning with Continuous Resources in Stochastic Domains ». In : p. 1049–1055.

MARIS, F. et P. RÉGNIER (2008). « TLP-GP : New Results on Temporally-Expressive Planning Benchmarks ». In : *Proceedings of the International Conference on Tools with Artificial Intelligence*, p. 507–514.

MCDERMOTT, D. (2005). *OPT Manual Version 1.7.3*.

MELAX, S. (1993). « New approaches to moving target search ». In : *Proceedings of AAAI Falls Symp. Game Planning Learn*, p. 30–38.

MILLER, G. (1996). « Hubble Space Telescope Planning and Scheduling – Experience, Lessons Learned and Future Directions ». In : *New observing modes for the next century* 87, p. 158–161.

MURRIETA-CID, R. et al. (2007). « Surveillance Strategies for a Pursuer with Finite Sensor Range ». In : *International Journal of Robotics Research* 26.3, p. 233–253.

MYERS, K. et al. (2002). « PASSAT : A user-centric planning framework ». In : *Proceedings of the International NASA Workshop on Planning and Scheduling for Space*, p. 1–10.

NAKHOST, H. et M. MÜLLER (2009). « Monte-Carlo Exploration for Deterministic Planning ». In : *Proceedings of tne International Joint Conference on Artificial Intelligence*, p. 1766–1771.

NAU, D. et al. (1999). « SHOP : Simple hierarchical ordered planner ». In : *Proceedings of the international Joint Conference on Artificial Intelligence*, p. 968–973.

NAU, D. et al. (2003). « SHOP2 : An HTN Planning System ». In : *Journal of Artificial Intelligence Research* 20.1, p. 379–404.

NEBEL, B. et J. KOEHLER (1995). « Plan Reuse versus Plan Generation : A Theoretical and Empirical Analysis ». In : *Artificial Intelligence* 76, p. 427–454.

NEWTON, M. et al. (2007). « Learning Macro-Actions for Arbitrary Planners and Domains ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 256–263.

NGUYEN, X., S. KAMBHAMPATI et R. NIGENDA (2002). « Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search ». In : *Artificial Intelligence* 135.1-2, p. 73–123.

NISSIM, R. et R. BRAFMAN (2012). « Multi-agent A* for parallel and distributed systems ». In : *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, p. 1265–1266.

— (2014). « Distributed Heuristic Forward Search for Multi-agent Planning ». In : *Journal of Artificial Intelligence Research* 51, p. 293–332.

PARSONS, T. D. (1976). « Pursuit-evasion in a graph ». In : *Theory and Application of Graphs*, p. 426–441.

PEDNAULT, E. (1994). « ADL and the State-Transition Model of Action ». In : *Journal of Logic and Computation* 4.5, p. 467–512.

PELLIER, D. (2010). « Distributed Planning through Graph Merging ». In : *Proceedings of the International Conference on Agents and Artificial Intelligence*, p. 128–134.

PELLIER, D., B. BOUZY et M. MÉTIVIER (2010a). « A mean-based approach for real-time planning ». In : *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, p. 1395–1396.

— (2010b). « An UCT Approach for Anytime Agent-Based Planning ». In : *Proceedings of the International Conference on Advances in Practical Applications of Agents and Multiagent Systems*, p. 211–220.
— (2011). « Using a Goal-Agenda and Committed Actions in Real-Time Planning ». In : *Proceedings of the International Conference on Tools with Artificial Intelligence*, p. 74–81.
PELLIER, D. et H. FIORINO (2005a). « Coordinated exploration of unknown labyrinthine environments applied to the pursuit evasion problem ». In : *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, p. 895–902.
— (2005b). « Multi-Agent Assumption-Based Planning ». In : *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, p. 1717–1718.
— (2007). « A Unified Framework Based on HTN and POP Approaches for Multi-Agent Planning ». In : *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, p. 285–288.
— (2009). « Un modèle de composition automatique et distribuée de services web par planification ». In : *Revue d'Intelligence Artificielle* 23.1, p. 13–46.
— (2017). In : *Journal of Experimental & Theoretical Artificial Intelligence*.
PELLIER, D., H. FIORINO et M. MÉTIVIER (2013). « Planifier lorsque le but change. Une approche inspirée de la recherche de cible mouvante ». In : *Revue d'Intelligence Artificielle* 27.2, p. 217–242.
— (2014). « Planning When Goals Change : A Moving Target Search Approach ». In : *Proceedings of International Conference on the Advances in Practical Applications of Heterogeneous Multi-Agent Systems*, p. 231–243.
PENBERTHY, J. et D. WELD (1992). « UCPOP : A sound, complete, partial order planner for ADL ». In : *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, p. 103–114.
PISTORE, M. et P. TRAVERSO (2001). « Planning as model checking for extended goals in non-deterministic domains ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 479–484.
POHL, I. (1970). « Heuristic search viewed as path finding in a graph ». In : *Artificial Intelligence*, p. 193–204.
PRUD'HOMME, C., J-G FAGES et X. LORCA (2014). *Choco3 Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S.
QUIGLEY, M., B. GERKEY et W. SMART (2015). *Programming Robots with ROS*. O'Reilly.
RAMANA, M. et M. KOTHARI (2017). « Pursuit-Evasion Games of High Speed Evader ». In : *International Journal of Robotics Research* 85.2, p. 293–306.
RAMIREZ, M., N. LIPOVETZKY et C. MUISE (2015). *Lightweight Automated Planning ToolKiT*. http://lapkt.org/.
RAMOUL, A. et al. (2016). « HTN Planning Approach Using Fully Instantiated Problems ». In : *Proceedings of the International Conference on Tools with Artificial Intelligence*, p. 113–120.
— (2017). « Grounding of HTN Planning Domain ». In : *International Journal on Artificial Intelligence* 26.5, p. 113–120.
RICHTER, S., M. HELMET et M. WESTPHAL (2008). « Landmarks Revisited ». In : *Proceedings of the Association for the Advancement of Artificial Intelligence*, p. 975–982.
RICHTER, S. et M. WESTPHAL (2008). « The LAMA Planner Using Landmark Counting in Heuristic Search ». In : *Proceedings of the International Conference on Planning and Scheduling*.

RINTANEN, J. (2012). « Planning as satisfiability : Heuristics ». In : *Artificial Intelligence* 193, p. 45–86.

— (2014). « Madagascar : Scalable Planning with SAT ». In : *Proceedings of the International Competition of Planning*.

RINTANEN, J., K. HELJANKO et I. NIEMELÄ (2006). « Planning as satisfiability : parallel plans and algorithms for plan search ». In : *Artificial Intelligence* 170.12–13, p. 1031–1080.

ROBINSON, N. et al. (2009). « SAT-based parallel planning using a split representation of actions ». In : *Proceedings of the International on Automated Planning and Scheduling*, p. 281–288.

SACERDOTI, E. (1975). « The nonlinear nature of plans ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 206–214.

SALNITRI, M., E. PAJA et P. GIORGINI (2015). *Maintaining Secure Business Processes in Light of Socio-Technical System's Evaluation*. Rapp. tech. DISI-University of Trento.

SANNER, S. (2010). « Relational Dynamic Influence Diagram Language (RDDL) : Language Description ». Unpublished manuscript from the International Planning Competition website.

SHAH, M. et al. (2013). « Knowledge engineering tools in planning : state-of-the-art and future challenges. » In : *Proceedings of the ICAPS workshop on Knowledge Engineering for Planning and scheduling*.

SHIVASHANKAR, V. et al. (2013). « The GoDeL Planning System : A More Perfect Union of Domain-Independent and Hierarchical Planning ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 2380–2386.

SHOHAM, Y. et M. TENNENHOLTZ (1995). « On social laws for artificial agents societies : off-line design ». In : *Artificial Intelligence* 73.1–2, p. 231–252.

SHUE, L. et R. ZAMANI (1993). « An Admissible Heuristic Search Algorithm ». In : *Methodologies for Intelligent Systems*. LNAI 689. Springer.

SOHRABI, S., J. BAIER et S. MCILRAITH (2009). « HTN Planning with Preferences ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1790–1797.

SRIDHARAN, M., J. WYATT et R. DEARDEN (2008). « HiPPo : Hierarchical POMDPs for Planning Information Processing and Sensing Actions on a Robot ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 346–354.

STENZ, A. (1995). « The focused D* algorithm for real-time replanning ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1642–1659.

STOLBA, M., D. FISER et A. KOMENDA (2015). « Admissible Landmark Heuristic for Multi-Agent Planning ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 211–219.

— (2016). « Potential Heuristics for Multi-Agent Planning ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 308–316.

STRENZKE, R. et A. SCHULTE (2011). « The MMP : A Mixed-Initiative Mission Planning System for the Multi-Aircraft Domain ». In : *Proceedings of the International Conference on Automated Planning and Scheduling*, p. 74–82.

SUN, X., S. KOENIG et W. YEOH (2008). « Generalized Adaptive A* ». In : *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, p. 469–476.

SUN, X., W. YEOH et S. KOENIG (2009). « Efficient Incremental Search for Moving Target Search ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 615–620.

— (2010a). « Generalized Fringe-Retrieving A* : Faster Moving Target Search on State Lattices ». In : *Proceedings of the International Joint Conference on Autonous Agents and MultiAgent Systems*. T. 1081–1088.
— (2010b). « Moving Target D* Lite ». In : *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems*, p. 67–74.
SUZUKI, I. et M. YAMASHITA (1992). « Searching for mobile intruders in a polygonal region ». In : *SIAM Journal on Computing* 21(5), p. 863–888.
TAMBE, M. et H. JUNG (1999). « The Benefits of Arguing in a Team ». In : *Artificial Intelligence Magazine* 20.4, p. 85–92.
TATE, A. (1977). « Generating project networks ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 888–893.
TATE, A., B. DRABBLE et R. KIRBY (1994). « O-Plan2 : an open architecture for command, planning and control ». In : *Intelligent Scheduling*, p. 213–239.
TOKEKAR P. andVander Hook, J., D. MULLA et V. ISLER (2016). « Sensor Planning for a Symbiotic UAV and UGV System for Precision Agriculture ». In : *IEEE Trans. Robotics* 32.6, p. 1498–1511.
TONINO, H. et al. (2002). « Plan Coordination by Revision in Collective Agent-Based Systems ». In : *Artificial Intelligence* 142.2, p. 121–145.
TOVAR, B. et S. LAVALLE (2006). « Visibility-based pursuit-evasion with bounded speed ». In : *Proceedings the Workshop on Algorithmic Foundations of Robotic*.
TOZICKA, J. et al. (2014). « Multiagent Planning by Iterative Negotiation over Distributed Planning Graphs ». In : *Proceedings of the Workshop on Distributed and Mutiltiagent Planning (ICAPS)*, p. 7–15.
VON-MARTIAL, F. (1992). *Coordinating plans for autonomous agents*. Springer Verlag.
WESER, M., D. OFF et J. ZHANG (2010). « HTN robot planning in partially observable dynamic environments ». In : *Proceedings of the IEEE International Conference on Robotics and Automation*, p. 1505–1510.
WILKINS, D. (1984). « Domain-independent planning Representation and plan generation ». In : *Artifical Intelligence* 22.3, p. 269–301.
— (1990). « Can AI planners solve practical problems ? » In : *Computational Intelligence Journal* 6.4, p. 232–246.
WILLIAMS, B. et P. NAYAK (1997). « A reactive planner for a model-based executive ». In : *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 150–159.
W.JOHAL et al. (2015). « A Cognitive and Affective Architecture for Social Human-Robot Interaction ». In : *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, p. 71–72.
YOUNES, H. et M. LITTMAN (2004). *PPDDL 1.0 : an extension to PDDL for expressing planning domains with probabilistic effects*. Rapp. tech. CMU-CS-04-167. Carnegie Mellon University.
YOUNES, H. et R. SIMMONS (2003). « VHPOP : Versatile Heuristic Partial Order Planner ». In : *Journal of Artificial Intelligence Research* 20.1, p. 405–430.
ZAMAN, S. et al. (2013). « An integrated model-based diagnosis and repair architecture for ROS-based robot systems ». In : *Proceedings of the International Conference on Robotics and Automation*, p. 482–489.
ZHUO, H. et Q. YANG (2014). « Action-model acquisition for planning via transfer learning ». In : *Artificial Intelligence* 212, p. 80–103.
ZLOTKIN, G. et J. ROSENSCHEIN (1990). « Negotiation and Conflict Resolution in Non-Cooperative Domains ». In : *Proceedings of the American National Conference on Artificial Intelligence*, p. 100–105.