Planification multirobot pour des missions de surveillance avec contraintes de communication

Patrick Bechon

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)*

Présentée et soutenue le *26/05/2016* **par :**

PATRICK BECHON

Planification multirobot pour des missions de surveillance avec contraintes de communication

---

JURY

MAGALI BARBIER	Ingénieur de recherche	Directrice de thèse
NOURY BOURAQADI	Professeur d'université	Rapporteur
FRANÇOIS CHARPILLET	Directeur de recherche	Rapporteur
MALIK GHALLAB	Directeur de recherche	Membre du Jury
SIMON LACROIX	Directeur de recherche	Directeur de thèse
CHARLES LESIRE	Ingénieur de recherche	Co-encadrant
VÉRONIQUE SERFATY	Responsable I2R à la DGA	Membre du Jury
OLIVIER SIMONIN	Professeur d'université	Membre du Jury

École doctorale et spécialité :
EDSYS : Robotique 4200046
Unité de Recherche :
Département Commande des Systèmes et Dynamique du vol (DCSD) - ONERA
Directeurs de Thèse :
Magali BARBIER (ONERA) et *Simon LACROIX (LAAS)*
Rapporteurs :
Noury BOURAQADI et *François CHARPILLET*

À mes grands-parents,
qui m'ont tous donné, chacun à leur façon, le goût des sciences.

Remerciements

Ce travail a beau être considéré comme le mien, il est le résultat d'un tout qui englobe mon travail personnel tout en étant beaucoup plus grand. *Nous sommes des nains sur des épaules de géants*, et je ne serais rien sans mes géants. Les géants scientifiques sur lesquels repose ce travail sont cités à la fin de ce manuscrit, dans la section Bibliographie. Même si ce manuscrit résume trois années de travail scientifique il résume aussi trois années de bonne ambiance, à la fois professionnelle et personnelle, et trois années d'une aventure humaine qui m'ont profondément marqué. Les géants qui ont permis cela ont donc tout autant leur place dans ce manuscrit, et cette partie leur est dédiée.

Je voudrais tout d'abord commencer par remercier toutes les personnes sans qui cette thèse n'aurait pas eu lieu dans d'aussi bonnes conditions scientifiques.

En premier lieu mes encadrants :

- **Magali Barbier** : Merci pour ta disponibilité permanente et pour ton accompagnement tout au long de cette thèse. Je sais que je t'ai rappelé dès le début ton propre directeur de thèse et si je me suis senti aussi bien à l'ONERA tu en es la principale responsable. J'espère seulement que je ne t'ai pas fait manger trop de chocolats.

- **Charles Lesire** : Merci pour ton suivi régulier et pour la qualité de tes conseils techniques, à la fois pointus et de qualité. Sans toi, les manip d'ACTION n'auraient pu avoir lieu et cette thèse n'aurait sûrement pas été aussi fournie. Et finalement, merci d'avoir gardé le secret du Père Noël.

- **Simon Lacroix** : Merci pour ton accueil et ton soutien. Même sans supervision au jour le jour, j'ai apprécié ton accueil chaleureux dans l'équipe d'ACTION et du LAAS. Et j'ai aussi beaucoup apprécié nos discussions opportunistes, lorsque les circonstances nous rassemblaient.

- **Guillaume Infantes** & **Vincent Vidal** : Merci pour vos conseils et votre aide pour la partie planification de cette thèse.

Merci aux membres du jury **Véronique Serfaty**, **Olivier Simonin**, au président **Malik Ghallab** et aux rapporteurs **Noury Bouraqadi** et **François Charpillet** pour avoir examiné ce travail et pour leurs questions lors de la soutenance.

Merci à tous les membres de l'équipe d'ACTION sans qui ce travail n'aurait pas eu la portée qu'il a aujourd'hui. Un grand merci à **Christophe Grand** pour ton aide lors de l'intégration technique de mes travaux et pour tes compétences techniques en déploiement de robots réels qui ont été indispensables. Merci à l'équipe du LAAS avec **Pierrick Koch** et **Baptiste Crusson**. Merci à l'équipe RESSAC de l'ONERA : **Pierre Escalas**, **Alexandre Amiez** et **Augustin Manecy**. Et j'ai finalement une pensée pour **Paul Chavent**, qui n'a pas pu voir le dénouement de ce projet.

Une thèse ne se faisant pas uniquement sur un projet mais aussi dans un laboratoire donné, le cadre de travail a beaucoup contribué à la bonne ambiance et aux très bons souvenirs que je garde de ces trois années.

Tout d'abord le clan (ou dirais-je même le gang) des doctorants et stagiaires. La thèse a ceci de particulier qu'elle ne dure que 3 ans et mène rarement à une embauche au sein du même laboratoire après : les effectifs tournent (trop) vite. Un merci tout particulier à **Frâncis** et à **Jââââââââââcques** pour l'ambiance exceptionnelle dans le bureau. Merci pour les heures qu'on a passées à parler jeux vidéo, jeux de plateau, KSP, Chalk, forge et casques de réalité virtuelle. Merci à **Simon** pour l'accueil et les conseils techniques qui ont transformé un béotien complet en un apprenti administrateur système qui n'a plus peur d'une ligne de commande. Merci à **Pierre** qui a su casser la barrière discret/continu pour organiser et participer à des soirée inter-UR de grande qualité. Merci à **Guillaume** pour avoir accepté de négocier avec moi l'API de libSTN afin que je puisse utiliser son travail. Merci à **Adrien** pour nos discussions passionnées d'un niveau « là-haut quoi ». Avec ta nouvelle formation, je ne serai sans doute plus au niveau de suivre ... Merci à **Emmanuel** pour avoir repris le flambeau des diplômes et cadeaux de thèse. Merci aussi à **Jérémy**, **Gabiche**, **Mathieu**, **Marine**, **Hélène**, **Igor**, **Jorrit**, **Victor** et **Nicolas D**.

Mais un laboratoire n'est pas uniquement défini par ses doctorants, et j'ai eu la chance de pouvoir aussi profiter des permanents, entre autre dans la salle de pause commune. Merci à **Stéphanie**, **Cédric Pralet**, l'autre **Charles**, **Cédric Seren**, **Jean-Loup**, **Catherine**, **Florent** et **Xavier**.

Et merci à tous pour vos cadeaux de départ.

Merci à la **SNCF** de m'avoir fourni environ 10h de rédaction par semaine. Et finalement, un merci à **Delphine Dufourd-Moretti** pour ses conseils d'orientation professionnelle qui m'ont conduit au PEA ACTION.

Au-delà du milieu professionnel, beaucoup d'autres personnes ont contribué indirectement au succès de cette thèse.

Tout d'abord ma famille sans qui, pour respecter un poncif du genre, je n'aurais pas pu être là. Merci à vous tous (**Papa**, **Maman**, **Guillaume**, **NatNat**) pour tout. Essayer de faire une liste ici de ce que vous avez fait pour moi ne pourrait être qu'incomplet et réducteur, je m'en abstiendrai donc, mais je n'en pense pas moins. Merci à **Papy** et à **Mamie** pour leur soutien et pour avoir utilisé tous les prétextes possibles pour venir me voir et m'aider dans ce que j'entreprends. Merci à **Céline**, **Cathy**, **Yona**, **Thierry** et **Raphaël** pour leur accueil au début et à la fin de ma thèse, ainsi que pour leur soutien le jour J.

Merci à mes amis toulousains : **Sergio** pour son initiation à la pêche et ses combats contre les forces de l'Empereur avec **Nghia**, **Pascal**, **Angela** et **Thibault**. Merci à **Fabien**, **Quentin** et **Loïc** pour notre partie mémorable de Risk Legacy. Et souvenez-vous, maintenant que les Aliens ont débarqué, plus rien ne sera comme avant.

Merci aussi à mes amis plus lointains géographiquement. Merci à **Hélène** et **Romain** pour leur sacrifice providentiel et pour entretenir la flamme du JdR. Merci à **Delphine**, **Flavien**, **Thomas**, **Guillaume**, **Anaïs** et **Steven**, et je souhaite sincèrement que malgré la distance nous allons pouvoir continuer de garder contact.

Et finalement, le meilleur pour la fin, la personne qui est à la fois mon amie la plus chère et le membre de ma famille la plus proche : **Servane**. Je ne peux sans doute pas écrire ici plus que ce que je t'ai déjà dit, mais sache que tu as joué un rôle important dans

la réussite de cette thèse et de bien plus encore par ton soutien.

Dans l'éventualité très probable où j'aurais malencontreusement oublié quelqu'un, merci de m'aider une dernière fois dans mon travail pour réparer cet oubli regrettable.

Je remercie _____ pour :

☐ son soutien indéfectible ;

☐ ses conseils avisés ;

☐ sa gentillesse et sa sincérité ;

☐ sa bonne humeur ;

☐ autre (précisez) : _____.

Pour finir, je tiens à signaler les compliments qui m'ont le plus touché, la plupart étant dû à plusieurs personnes qui se reconnaitront. Ce sont les plus beaux compliments que vous pouviez me faire, et je vous en remercie du plus profond de mon cœur.

Ta soutenance était claire, j'ai tout compris alors que je ne suis pas du domaine.

Tu as été un acteur majeur des démonstrations du PEA ACTION et tu t'es très bien intégré dans l'équipe.

Ça a été un plaisir de travailler avec toi pendant ces trois années, tu vas nous manquer.

Un dernier merci tout particulier à leurs auteurs.

TGV entre Toulouse et Tours, le 27 mai 2016.

Table des matières

Liste des figures

Liste des acronymes

AAV Autonomous Aerial Vehicle : véhicule aérien autonome.

AGV Autonomous Ground Vehicle : véhicule terrestre autonome.

ANML Action Notation Modeling Language (cf. section 1.1.7).

AXV Autonomous Vehicle : véhicule autonome (AAV ou AGV).

DGA Direction Générale de l'Armement.

HiPOP Hierarchical Partial Order Planner : le planificateur développé dans ces travaux.

HTN Hierarchical Task Network (cf. section 1.3.1).

IPC International Planning Competition (cf. `http://icaps-conference.org/index.php/Main/Competitions`).

ISMAC Interface de Supervision de Mission pour ACtion.

LAAS Laboratoire d'Analyse et d'Architecture des Systèmes : laboratoire du CNRS.

MaSTN Multiagent Simple Temporal Network.

METAL Mastn Execution with TemporAl fLexibility : le superviseur développé dans ces travaux.

MORSE Modular OpenRobots Simulation Engine : Simulateur robotique open source (cf. `https://www.openrobots.org/morse`).

ONERA Office National d'Études et de Recherches Aérospatiales.

PDDL Planning Domain Description Language (cf. section 1.1.3).

PEA Programme d'Études Amont de la DGA.

POP Partial Order Plannning (cf. section 1.2.3).

ROS Robot Operating System (cf. `http://www.ros.org/`).

SIMATO Specification Interface for Multi-Agents Tactical Operations.

ssi. si et seulement si.

STN Simple Temporal Network (cf. page 33).

TFD Temporal Fast Downward : planificateur utilisé dans ces travaux.

VHPOP Versatile Heuristic Partial Order Planner : planificateur utilisé dans ces travaux.

YAHSP Yet Another Heuristic Search Planner : planificateur utilisé dans ces travaux.

Notations

f	un fluent
Π	un plan
P	un problème
a	une action
τ	une tâche
$\mathcal{T}$	un ensemble de tâches
m	une méthode
s	un état
t	une date ou un instant temporel
d	une durée
ε	un événement
δ	un défaut
Δ	un défaut
ϕ	une étiquette temporelle
h	l'horizon de planification
$\mathcal{P}$	une famille de fluent de position (voir chapitre 4)
Ω	ensemble des dernières positions possibles (voir chapitre 4)

Introduction

De nombreux travaux ont porté sur l'autonomie individuelle des robots mobiles définie par leur capacité à réaliser une mission avec peu de contacts avec les opérateurs humains. Depuis les premières expérimentations sur l'autonomie décisionnelle à la fin des années 60 [Nilsson 1984], des progrès importants ont été réalisés. L'étape suivante dans le développement de cette autonomie individuelle est le passage à l'autonomie collective d'une équipe de robots pour accomplir ensemble une mission.

Des travaux existent déjà sur la réalisation d'une mission complexe par une équipe constituée de robots autonomes. La réalisation d'une telle mission nécessite de prendre en compte explicitement les synergies entre robots ainsi que leurs différences. Les buts ne sont atteignables qu'en faisant collaborer plusieurs robots et les critères numériques à optimiser comme la durée ou la consommation énergétique nécessitent des interactions poussées entre robots durant la mission. Cette coopération poussée est ainsi nécessaire pour la réalisation de missions de recherche et sauvetage, de localisation d'objets d'intérêt, de surveillance de zones, etc. Un exemple est présenté sur la figure 1. Le point commun de ces missions est de devoir adapter le comportement des robots pendant la réalisation de la mission : l'environnement est contraignant, les actions des robots sont complexes et non déterministes et les perturbations remettant en cause le plan des robots ne sont pas rares.

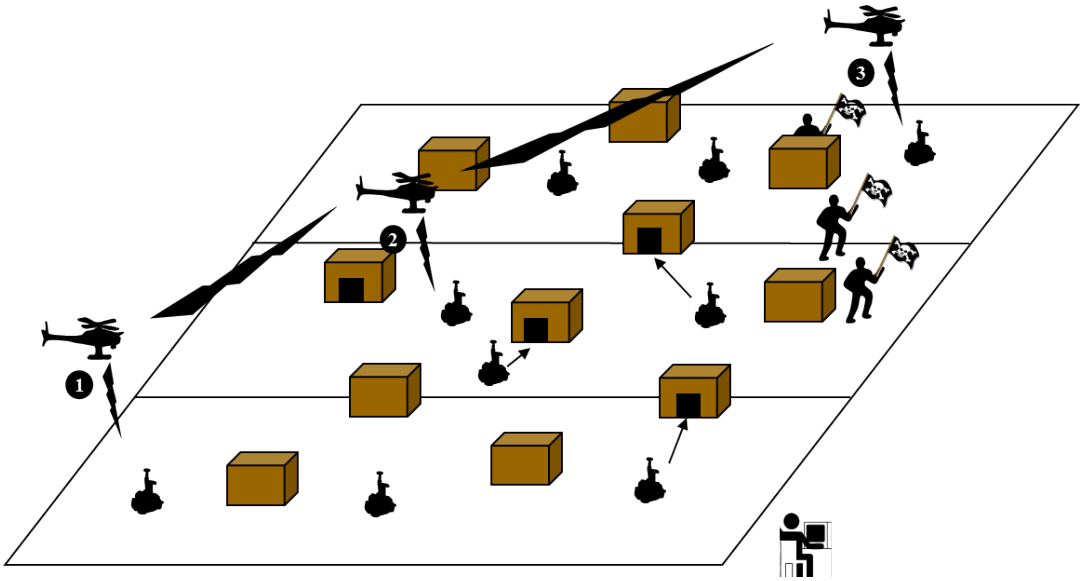


Figure 1 – Exemple d'un scénario de surveillance de zone. Les robots sont répartis en sous-équipes, chacune en charge d'une sous-zone. Des communications à l'intérieur et entre sous-équipes sont nécessaires pour localiser et poursuivre des cibles présentes dans la zone.

Gateau et al. ont proposé une architecture décisionnelle pour une équipe de robots hétérogènes. La supervision de la mission par HiDDEN [Gateau, Lesire et Barbier 2013]

est basée sur l'exécution et la réparation en ligne d'un plan hiérarchique d'actions (utilisant le formalisme HTN, Hierarchical Task Network). Des expérimentations avec trois robots de nature différente ont mis en avant la capacité de réagir à des événements imprévus. Cette architecture utilise cependant des plans ne prenant pas en compte l'aspect temporel, c'est à dire qu'ils ne permettent pas de détecter l'absence d'un robot à un rendez-vous, un retard trop important, etc. De plus la réparation n'est possible qu'en suivant des procédures ad hoc.

Fernández-Olivares et al. ont proposé une architecture décisionnelle pour la coordination d'une équipe de secours dans un contexte de gestion de crise. SIADEX [Fernández-Olivares et al. 2006] est basée sur des HTN intégrant une notion de temps. Cette architecture produit des plans flexibles temporellement et intégrant des contraintes temporelles riches. Par contre elle nécessite l'introduction d'expertise ad-hoc et n'a pas été utilisée pour une équipe de robots, uniquement pour coordonner des équipes humaines.

Carlési et al. ont proposé une architecture de contrôle pour une flottille d'engins sous-marins autonomes et hétérogènes. REMORAS [Carlési et al. 2011] repose sur une organisation des robots en équipe pour permettre de s'abstraire des différences entres les agents et permettre leur collaboration. Les mécanismes de coopération sont adaptés aux contraintes du milieu sous-marin qui limitent très fortement les communications.

Ces exemples de travaux montrent l'intérêt et la faisabilité de missions multiagents exploitant les différences de capacités entre ces agents. Néanmoins ces architectures ne permettent pas de gérer une large gamme d'aléas et les communications sont souvent considérées comme étant garanties ou extrêmement limitées. De plus, la plupart d'entre elles nécessitent souvent une expertise humaine poussée.

C'est ce qui justifie cette étude. Elle a consisté à créer et à valider une architecture distribuée intégrant planification, supervision de l'exécution du plan et réparation de ce plan suite à un aléa pour une équipe de robots hétérogènes lui permettant d'effectuer sa mission en environnement réel et sous contraintes de communication.

La démarche a consisté dans un premier temps à se concentrer sur la fonction planification de cette architecture. Les communications non garanties ont orienté nos recherches vers une intégration de la décision à bord de tous les robots et l'introduction de rendez-vous entre les robots pour garantir une synchronisation en cours de mission. Nous avons d'abord choisi un type de modélisation du problème et un type de raisonnement pour la planification. Ce choix a été fait pour pouvoir gérer des objectifs avec des échéances temporelles, pour permettre la gestion des rendez-vous, et faciliter la réparation des plans en cours d'exécution. Pour cela, les plans produits doivent être enrichis par rapport à une séquence d'actions rigide temporellement. Cela nous a mené à choisir la planification hybride, dont le principe est d'utiliser des actions hiérarchiques lors d'une planification POP (*Partial Order Planning*). Nous avons ensuite implémenté un algorithme de ce type nommé HiPOP. Cela a nécessité de définir un modèle précis pour les actions hiérarchiques : ce modèle a été construit pour permettre l'ajout opportuniste d'actions hiérarchiques durant la planification afin d'améliorer les performances d'HiPOP. Pour lui permettre d'utiliser au mieux ces actions hiérarchiques, nous avons choisi des heuristiques guidant la recherche adaptées à cette modélisation. Afin de vérifier la pertinence de ces choix, nous avons testé HiPOP sur un ensemble de benchmarks. Ces benchmarks sont partiellement issus des compétitions IPC (International Planning Competition) et ont été sélectionnés car ils permettent de définir une hiérarchie d'actions et de couvrir un large éventail de problèmes. (Chapitre 3).

Ce travail a fait l'objet d'une publication intitulée « HiPOP : Hierarchical Partial-Order Planning » [Bechon et al. 2014].

Des limites apparues durant cette validation expérimentale en simulation nous ont conduit à étudier plus précisément l'influence du raisonnement géométrique. Un tel raisonnement est obligatoire dans des problèmes de robotique mobile et il possède des spécificités qui peuvent être exploitées lors de la planification : un robot ne peut être qu'à une unique position à tout moment. Une analyse du domaine permet de détecter ces positions et d'adapter la recherche pour tirer partie de cette propriété. En particulier, on peut plus facilement évaluer la difficulté d'assurer qu'un robot sera à une position donnée en fonction du plan en cours de construction et non pas uniquement en fonction de l'état initial. Cette propriété permet aussi de différer la résolution des défauts portant sur ces positions en introduisant de nouvelles contraintes temporelles. Afin de mesurer l'impact de ces modifications sur la taille des problèmes résolus, des tests ont été lancés sur les mêmes benchmarks que précédemment (Chapitre 4).

HiPOP peut être utilisé pour calculer un plan initial, avant le lancement d'une mission par une équipe de robots. Mais pendant l'exécution de cette mission, les aléas qui surviennent doivent être surmontés en réparant le plan en cours d'exécution. Cela nécessite de produire rapidement des nouveaux plans contenant impérativement les actions déjà exécutées. Pour cela HiPOP a été adapté pour utiliser son algorithme de recherche dans un processus de réparation. Plus précisément, le plan de départ de la recherche est modifiée et la recherche procède toujours par ajout itératif d'éléments. Ce plan initial doit contenir les actions passées mais contient aussi les actions futures qui restent pertinentes malgré l'aléa rencontré. Cela permet de limiter la recherche effectuée et de produire un plan proche du plan initial. La présence d'actions hiérarchiques dans le plan est utilisée au moment du choix des éléments du plan à retirer. Une validation expérimentale sur un ensemble de benchmarks de réparation a été faite. Les critères étudiés ont été la vitesse de calcul de plans ainsi que la stabilité du plan (Chapitre 5). Ce travail a fait l'objet d'une publication intitulée « Using hybrid planning for plan reparation » [Bechon et al. 2015].

Une fois cette extension réalisée, HiPOP était prêt à être utilisé au sein d'une architecture complète de contrôle d'une équipe de robots hétérogènes. La première étape a consisté à créer un algorithme d'exécution capable d'utiliser la flexibilité temporelle pour limiter le nombre d'appels au processus de réparation. Il a ensuite été nécessaire d'adapter cet algorithme pour lui permettre d'utiliser l'algorithme de planification pour réparer le plan courant. Pour le projet ACTION, nous avons rassemblé ces travaux dans une architecture nommée METAL. Nous avons testé cette architecture dans le cadre de simulations intégrant plusieurs robots hétérogènes. Puis cette architecture a été utilisée lors d'expérimentations en environnement réel mettant en œuvre simultanément jusqu'à 3 drones aériens et 9 robots terrestres (Chapitre 6).

Première partie

État de l'art

1

Planification

Sommaire

Un des premiers paradigmes utilisé en robotique a été celui de « Percevoir-Planifier-Agir » (*Sense-Plan-Act*, voir figure 1.1). Dans ce paradigme un robot perçoit d'abord son environnement et en construit une représentation interne. Puis un ensemble d'actions (appelé un plan) est calculé pour atteindre ses buts en utilisant cette représentation. Finalement, il peut exécuter ces actions.

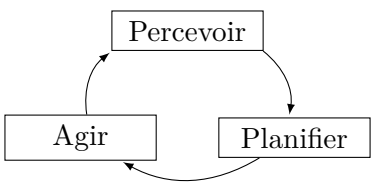


Figure 1.1 – Représentation du paradigme Percevoir-Planifier-Agir.

En pratique, cette approche doit être itérative : le robot doit continuer de percevoir son environnement pour s'assurer que les actions produisent bien les effets escomptés et pour pouvoir détecter une nouvelle situation éventuelle ayant invalidé le plan actuel. Dans une telle situation, il faut alors trouver un nouveau plan adapté en faisant de nouveau appel à la planification.

Même si d'autres approches ont été proposées depuis pour une intégration plus fine de ces trois fonctions, elles n'en restent pas moins des composantes nécessaires d'un système robotique.

Dans ce mémoire, nous nous intéresserons particulièrement à la planification et à son interaction avec les deux autres fonctions.

Un algorithme de planification peut être décrit comme un algorithme produisant un plan à partir des entrées suivantes :

- une description de l'état actuel du monde ;
- une liste d'actions disponibles ;
- un ou plusieurs buts à atteindre.

Un plan est alors défini comme un ensemble d'actions et de contraintes (d'ordonnancement temporel par exemple) entre ces actions. Ces contraintes permettent d'ordonner les actions entre elles pour assurer l'exécution du plan. Selon les cadres de modélisation utilisés, les buts peuvent être décrits différemment : un ensemble de propriétés à vérifier sur l'état du monde atteint après l'application du plan, une fonction à maximiser, une action particulière à instancier, etc.

Dans un contexte robotique, les entrées sont calculées en fonction de plusieurs éléments d'un modèle initial du monde et des perceptions du robot. Le modèle initial inclut les capacités des actionneurs, un modèle à priori de son environnement, etc. Les perceptions du robot permettent de mettre ce modèle à jour pour coller au mieux à la situation actuelle. Par exemple un obstacle repéré rend certaines actions de déplacement impossibles, un capteur GPS permet de mettre à jour la position du robot, etc. Ces entrées prennent souvent la forme d'une représentation symbolique et simplifiée des informations obtenues par les capteurs. Les actions du plan sont souvent symboliques aussi et nécessitent donc une interprétation par la partie « Agir ». Par exemple une action de déplacement ne permettra pas de connaître les commandes exactes à appliquer aux moteurs, mais la partie « Agir » sera chargée d'effectuer le déplacement du robot.

Exemple : Problème de planification

Dans la suite de ce manuscrit, on utilisera de manière récurrente un type de problème de planification particulier, présenté ici.
Le problème consiste à utiliser un ensemble de robots pour explorer une zone donnée. On suppose ici que les robots sont autonomes : il savent se rendre seul d'un point A à un point B, faire une observation seul, etc. Nous nous intéressons à la capacité d'autonomie de l'équipe. On considère alors deux types de robots : des robots terrestres (AGV - *autonomous ground robot*) et des robots aériens (AAV - *autonomous aerial robot*). On suppose que la zone est divisée en cellules arrangées sous forme de grille. Les cellules peuvent être étiquetées comme étant à observer ou déjà observées. De plus, chaque cellule peut avoir un terrain particulier.

- Un terrain vide où les deux types de robots peuvent circuler

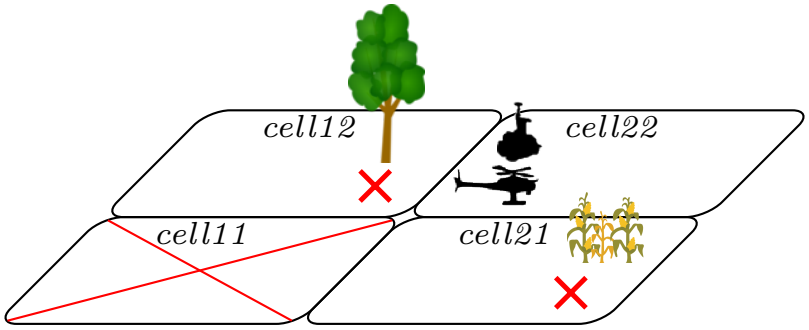
– Un champ qu'un AAV peut survoler et observer mais où un AGV ne peut pas pénétrer

– Une forêt dans laquelle un AGV peut se déplacer et l'observer mais un AAV ne peut pas la survoler (pour pouvoir utiliser un algorithme d'atterrissage automatique en cas de problème par exemple)

– Un terrain impénétrable où aucun robot ne peut pénétrer.

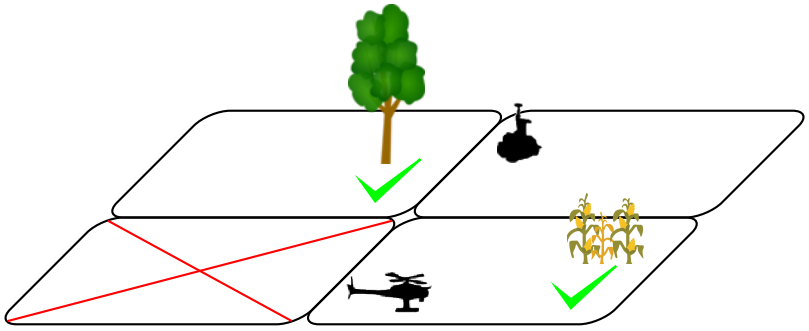
Le but du problème est de trouver une trajectoire pour chacun des robots permettant d'explorer toutes les cellules marquées comme devant être observées. Les actions disponibles pour chaque robot sont les déplacements (entre deux cases adjacentes) et les observations (de la cellule sur laquelle le robot se trouve).

Figure 1.2 – Exemple d'un problème de planification.



(a) Exemple de représentation d'un état initial possible.

La figure 1.2a montre un état initial possible pour un problème simple sur une grille 2x2. Deux robots sont présents : un AAV et un AGV. Ils sont au départ dans la cellule 22 qui est un terrain vide. La cellule 11 est impénétrable et les deux autres cellules sont une forêt et un champ à explorer. Une croix rouge dans coin inférieur droit indique une cellule qui doit être explorée.



(b) Exemple de représentation d'un état final possible.

La figure 1.2b présente un des états atteignables (après plusieurs actions) depuis l'état précédent. Les deux cellules objectifs ont été observées et on remarque que

l'AGV est revenu à sa position initiale. La marque verte dans le coin inférieur droit indique que la cellule a bien été explorée.

1.1 Cadre de modélisation

La première étape pour une algorithme de planification est de définir les données d'entrée sur lequel il agira : c'est le cadre de modélisation qu'il utilisera pour représenter le monde.

Plusieurs cadres existent dans la littérature. Ils se distinguent selon deux principaux critères.

- La puissance de représentation : tous les langages ne permettent pas de représenter les mêmes problèmes.
- La facilité d'utilisation : par exemple la quantité de travail nécessaire pour comprendre ou écrire un problème, la qualité de leur écosystème (nombre de planificateurs utilisant le même modèle, de problèmes ou d'outils disponibles), etc.

Un tableau récapitulatif des différentes descriptions présentées est donné page 27.

1.1.1 Description par état

La représentation sans doute la plus simple théoriquement pour la planification est une représentation du monde sous forme d'un graphe. Chaque état possible du monde est alors un nœud, et chaque arête représente une action qui fait passer d'un état à un autre.

Définition 1 : Planification comme une recherche dans un graphe

Un problème de planification est défini par :

- S un ensemble d'états ;
- A un ensemble d'actions ;
- $f : S \times A \to S$ une fonction partielle ;
- $I \in S$ l'état initial ;
- $S_G \subset S$: un ensemble d'états buts.

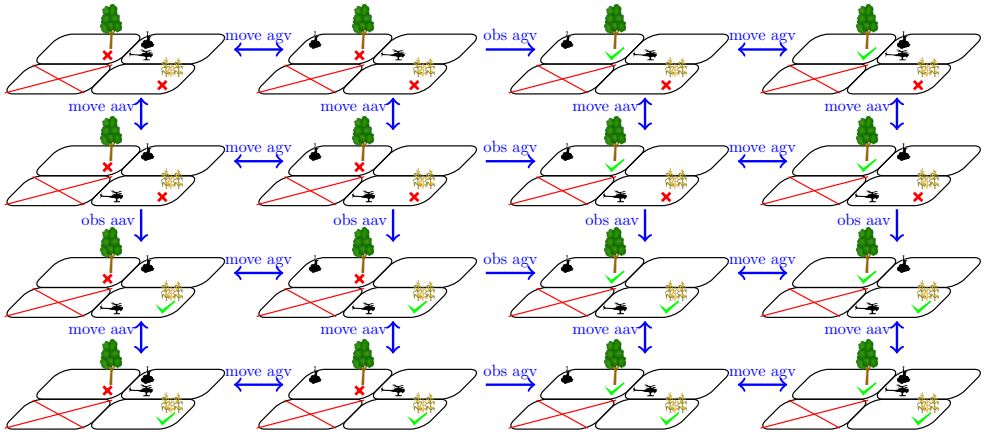
La fonction f permet de calculer l'état suivant étant donné l'état courant et l'action appliquée. Dans chaque état, toutes les actions ne sont pas forcement applicables et f n'est définie que pour les couples d'état et d'action compatibles. Elle permet donc de définir un graphe orienté dont l'ensemble des nœuds est S.

Le problème de planification revient alors à chercher un chemin dans ce graphe depuis I vers n'importe quel nœud de S_G.

Exemple : Planification comme une recherche dans un graphe

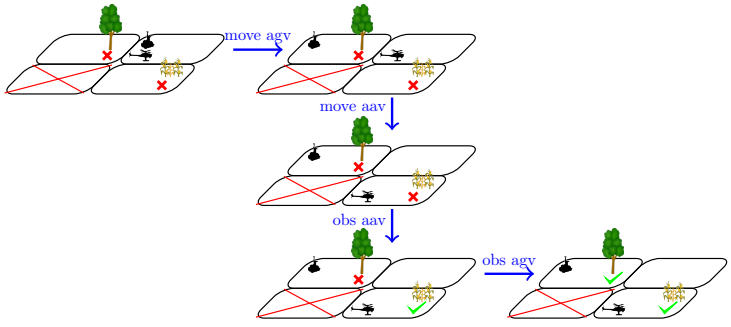
En considérant le problème défini par l'état initial présenté figure 1.2a, le graphe de tous les états atteignables est présenté figure 1.3a. Les actions disponibles sont les déplacements d'un robot (respectivement `move aav` et `move agv`) ainsi que l'observation par un robot d'une cellule à explorer (respectivement `obs aav` et `obs agv`).

Figure 1.3 – Exemple de planification comme recherche dans un graphe.



(a) Représentation sous forme de graphe d'un problème simple de planification.

L'état initial est en haut à gauche. Chaque flèche bleue correspond à l'application d'une action, qui fait passer d'un état à un autre état. Les actions de déplacement sont réversibles alors que les actions d'observation ne le sont pas. Les quatre états du quart en bas à droite correspondent aux quatre états satisfaisant le but : les deux lieux à observer le sont.
La figure 1.3b présente un plan extrait du graphe précédent et résolvant le problème.



(b) Exemple de plan extrait de la recherche dans le graphe.

Ce plan contient 4 actions. L'état initial est bien I et l'état final est dans S_G

Bien que simple théoriquement, l'explosion combinatoire rend ce modèle inadapté à une utilisation concrète. En effet le nombre d'états à représenter (et donc la taille du graphe à explorer) augmente exponentiellement avec le nombre d'objets dans le problème. Dans cet exemple, un objet est un robot ou une cellule.

1.1.2 STRIPS

Pour pallier ce problème d'explosion combinatoire, le formalisme STRIPS [Fikes et Nilsson 1972] utilise une représentation factorisée des états. Ce formalisme a été introduit par le premier planificateur robotique, appelé STRIPS (STanford Research Institute Problem Solver), développé pour le robot Shakey [Nilsson 1984]

L'idée est de ne pas représenter chaque état comme un tout mais de le séparer en différentes composantes : la position de chaque robot par exemple. Un état est alors défini comme un ensemble de *fluents* prenant une valeur vrai ou faux. Chaque fluent peut être vu comme une propriété qui est vérifiée ou non dans un état donné : par exemple « le robot r_1 est au point pt_1 » ou « l'acquisition a_1 a été réalisée ».

Définition 2 : Problème de planification (STRIPS)

Un problème de planification est défini par :

- F un ensemble de fluents ;
- A un ensemble d'actions ;
- $I \subseteq F$ un ensemble représentant l'état initial ;
- $G \subseteq F$ un ensemble représentant le but.

Définition 3 : Etat (représentation STRIPS)

Étant donné un problème de planification $\langle F, A, I, G\rangle$, un état est défini comme un sous-ensemble de F.

Définition 4 : Action (représentation STRIPS)

Soit F un ensemble de fluents, une action est définie par :

- *name* un nom unique dans le problème considéré ;
- $Pre \subseteq F$ un ensemble de préconditions ;
- $Add \subseteq F$ un ensemble d'effets ;
- $Del \subseteq F$ un ensemble d'effacements.

Un plan pouvant contenir plusieurs fois la même action, il est nécessaire de faire la distinction entre une action et une tâche. Une tâche correspond à une réalisation donnée d'une action : un plan est donc constitué d'un ensemble de tâches distinctes, chacune représentant une action particulière. Une action peut avoir plusieurs tâches correspondantes dans le plan. Quand il n'y a pas d'ambiguïté possible, on utilisera le terme d'action pour désigner une tâche instanciant cette action.

Une action $a = \langle Pre, Add, Del\rangle$ est applicable à un état e ssi. $e \subseteq Pre$ et produit alors l'état $a[e] = ((e \cap \neg Del) \cup Add)$.

L'objectif de la planification est alors de trouver une séquence ordonnée d'actions telle que chaque action soit applicable (c'est-à-dire que ses préconditions sont vérifiées au

moment de son exécution) et que l'état final contienne G. Formellement on peut définir un plan comme une séquence de tâches :

Définition 5 : Plan (représentation STRIPS)

Étant donné un problème de planification $\langle F, A, I, G\rangle$, un plan est défini comme une séquence de tâches $\langle \tau_0, \tau_1, ..., \tau_n\rangle$, chaque τ_i étant l'instanciation de l'action $a_i \in A$. Sa longueur est définie comme son nombre de tâches : ici $n + 1$.

Un plan produit une séquence d'états $\langle s_0, s_1, ...s_{n+1}\rangle$ définie par $s_0 = I$ et $\forall i > 0, s_{i+1} = a_i[s_i]$.

Définition 6 : Plan solution (représentation STRIPS)

Un plan est valide ssi. $\forall i, Pre(a_i) \subseteq s_i$.

Un plan est solution du problème $\langle F, A, I, G\rangle$ ssi. il est valide et $G \subseteq s_{n+1}$.

Au vu de la difficulté de résolution des problèmes ou pour respecter des contraintes imposées sur la solution, il est intéressant de se restreindre à des plans de longueur bornée. Étant donné un problème non borné, on peut définir sa restriction à un horizon de planification donné comme le même problème où l'on ne s'intéresse qu'à des solutions moins longues que cet horizon. Cela permet d'éliminer les plans trop longs lors de la recherche et de ne raisonner que sur un ensemble fini de plans potentiels.

Définition 7 : Horizon de planification

Étant donné un problème de planification P, on définit comme P_h sa restriction à des plans de longueur h au maximum : on cherche un plan solution de longueur inférieur ou égale à h et le problème n'a pas de solution si toutes les solutions de P ont une longueur supérieure à h.

h est appelé l'horizon de planification du problème P_h.

Exemple : Planification STRIPS

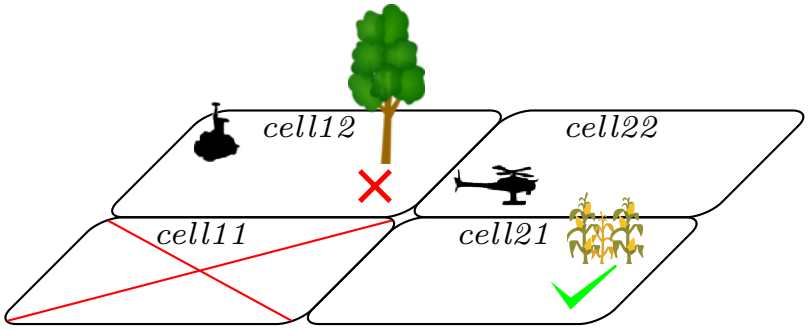


Figure 1.4 – Exemple d'état du monde et sa description en STRIPS.

On peut décrire l'état représenté à la figure 1.4 avec les fluents suivants.

- Pour décrire le terrain : `empty cell22`, `forest cell12`, `field cell21`, `blocked cell11`.
- Pour décrire la position des véhicules : `at agv cell12`, `at aav cell22`.
- Pour décrire l'état des cellules : `to-observe cell12`, `observed cell21`.

L'action de déplacement qui déplacerait l'AAV de la cellule `cell22` (vide) à `cell21` (le champ) serait décrite comme suit :

- *nom* : `move-aav-cell22-cell21`
- *Pre* = {`at aav cell22`}
- *Add* = {`at aav cell21`}
- *Del* = {`at aav cell22`}

L'intérêt de cette représentation est de permettre une description compacte des actions : au lieu de devoir décrire toutes les transitions possibles on décrit l'effet d'une action sur l'ensemble des états possibles à la fois. De ce point de vue STRIPS représente une nette amélioration en terme de praticité d'utilisation vis à vis de la description par un graphe. Par contre la puissance de représentation (*i.e.* l'ensemble des problèmes qui peuvent être représentés) est la même.

1.1.3 PDDL

PDDL (Planning Domain Definition Language) [McDermott et al. 1998] est une évolution du formalisme STRIPS initialement créée pour les compétitions internationales de planification. Ces compétitions sont organisées tous les deux ou trois ans depuis 1998 en même temps que les conférences ICAPS. PDDL utilise des variables (préfixées par ?) dans la description des actions ce qui permet une description compacte des problèmes. Les problèmes sont rassemblés en familles partageant les mêmes caractéristiques. Chaque

famille est décrite par un *domaine* qui contient des informations générales sur les actions disponibles et leurs effets en utilisant une description principalement sous forme de variables. Chaque *problème* est l'association d'un domaine avec une liste d'objets présents (permettant l'instanciation des variables du domaine), l'état initial et le but à atteindre. Cela permet par exemple d'avoir une représentation unifiée de tous les problèmes multirobots présentés dans les exemples de ce manuscrit : le domaine est le même pour tous et seuls le nombre de robots, le nombre de cellule, l'état initial et le but changent.

Exemple : Planification PDDL

En reprenant l'exemple présenté à la figure 1.2a, on peut construire une représentation en PDDL du problème.
La représentation complète est donnée dans l'annexe A, page 205. Une action de déplacement y est décrite dans le domaine comme suit.

```
(:action move-aav
  :parameters (?r - aav ?from ?to - loc)
  :precondition (and (at ?r ?from)
                     (adjacent ?from ?to)
                     (aav-allowed ?from)
                     (aav-allowed ?to))
  :effect (and (at ?r ?to) (not (at ?r ?from)) )
)
```

Elle prend en paramètre le nom de l'AAV qui doit bouger, son point de départ et son point d'arrivée. Elle n'est applicable que si l'AAV est au point de départ souhaité, si les deux lieux sont adjacents et si les deux lieux sont autorisés pour un AAV. Son effet est de déplacer l'AAV en créant un fluent et en effaçant le fluent devenu obsolète (l'ancienne position de l'AAV).
Le principal avantage de cette représentation est sa généralité : cette action suffit pour représenter tous les mouvements d'AAV quelque soit la disposition des cellules.
Le problème est alors défini par trois ensembles :

- la liste des objets existants (ici un AGV, un AAV et 4 cellules) ;

```
(:objects
  aav1 - aav
  agv1 - agv
  cell11 cell12 cell21 cell22 - loc
)
```

- l'état initial (défini par la liste des fluents disponibles) ;

```
(:init
  (adjacent cell12 cell22) (adjacent cell22 cell12)
  (adjacent cell21 cell22) (adjacent cell22 cell21)

  (aav-allowed cell21)
```

```
6   (agv-allowed cell12)
7   (aav-allowed cell22) (agv-allowed cell22)
8
9   (at aav1 cell22)
10  (at agv1 cell22)
11 )
```

– le but à atteindre (tout état où les deux lieux spécifiés sont explorés).

```
1 (:goal (and
2   (explored cell12)
3   (explored cell21)
4 ))
```

Les plans produits sont de la forme suivante :

```
0: (move-aav aav1 cell22 cell21) [1]
1: (move-agv agv1 cell22 cell12) [1]
2: (explore aav1 cell21) [1]
3: (explore agv1 cell12) [1]
```

Chaque ligne indique le numéro de l'action, son nom (instancié) et sa durée (ici définie par défaut à 1).

Ce langage étant assez répandu grâce aux compétitions des IPC, son écosystème est riche. Tous les planificateurs concourant aux IPC sont disponibles, ainsi que les définitions des problèmes utilisés. De plus, un vérificateur de plan, permettant de tester la validité d'un plan sur un domaine et un problème donné, est également disponible.

Plusieurs évolutions ont été proposées à PDDL : une représentation des problèmes temporels et de variables numériques [Fox et Long 2003], les prédicats dérivés et les conditions initiales temporelles [Edelkamp et Hoffmann 2004], des aspects multiagents [Kovacs 2012] ou encore la définition de métriques à utiliser pour évaluer la qualité d'un plan [Gerevini et Long 2006].

1.1.4 Représentation du temps dans PDDL

Une des limitations des trois cadres précédents est leur puissance de représentation limitée : les plans ne sont qu'une succession linéaire d'actions sans notion temporelle de durée des actions ou d'exécution concurrente. La version 2.1 de PDDL [Fox et Long 2003] propose une extension temporelle. Elle permet de doter les actions d'une durée et de faire référence aux dates de début et de fin de chaque action pour les préconditions et les effets. Cela permet d'imposer des actions en parallèle ou au contraire d'en interdire le chevauchement.

La définition d'un problème et d'un état est reprise de la formulation STRIPS (respectivement les définitions 2 et 3). Par contre les actions sont définies avec des contraintes exprimées par des *étiquettes temporelles*.

Définition 8 : Fluent temporel (PDDL2.1)

Soit F un ensemble de fluents.
Un fluent temporel est défini comme une paire $\langle f, \phi \rangle$ où $f \in F$ est un fluent et ϕ est une étiquette temporelle $\phi \in \{over\_all, at\_start, at\_end\}$.
On note $\tilde{F}$ l'ensemble des fluents temporels.

Définition 9 : Action (PDDL2.1)

Soit $\tilde{F}$ un ensemble de fluents temporels, une action est définie par :

- *name* un nom unique dans le problème considéré ;
- $d \in \mathbb{R}^+$ une durée ;
- $Pre \subseteq \tilde{F}$ un ensemble de préconditions ;
- $Add \subseteq \tilde{F}$ un ensemble d'effets ;
- $Del \subseteq \tilde{F}$ un ensemble d'effacements.

De plus, les étiquettes temporelles *over_all* ne peuvent être utilisées que pour les préconditions d'une action.

Les conditions d'application d'une action et ses effets dépendent donc des étiquettes temporelles. Une action n'est applicable que si toutes ses préconditions sont valides. Une précondition $\langle f, \phi \rangle$ est ainsi valide :

- Si $\phi = at\_start$ et que f est vrai au début de l'action
- Si $\phi = at\_end$ et que f est vrai à la fin de l'action
- Si $\phi = over\_all$ et que f est vrai sans discontinuer du début à la fin de l'action (ces deux points étant exclus de l'intervalle)

Un effet se réalise soit au début de la tâche (si $\phi = at\_start$) soit à la fin de la tâche (si $\phi = at\_end$).

Définition 10 : Plan (PDDL2.1)

Soit $\langle F, A, I, G \rangle$ un problème de planification.
Un plan est défini comme une séquence de tâches datées : $\mathcal{T} = \langle \tau_0, ...\tau_n \rangle$ où $\forall i, \tau_i = \langle t_i, a_i \rangle$, $a_i \in A$ et $t_i \in \mathbb{R}$ représente la date de début de τ_i.

Pour toute tâche s_i de durée d_i, si t_i est sa date de début alors $t_i + d_i$ est sa date de fin. Pour simplifier les notations, on notera alors t_i^i et t_i^f respectivement sa date de début et sa date de fin.

Les actions d'un plan pouvant se chevaucher, il est nécessaire de raisonner sur l'enchaînement des dates de début et de fin des actions. PDDL2.1 ne permet pas d'ajouter des contraintes ou de modifier l'état courant en dehors de ces points particuliers. On appelle alors événements l'ensemble des dates de début et de fin de toutes les tâches du plan.

Définition 11 : Événement (PDDL 2.1)

Soit $\langle F, A, I, G\rangle$ un problème de planification.
Soit $\Pi = \langle \langle t_0, \tau_0\rangle, ... \langle t_n, \tau_n\rangle\rangle$ un plan.
On note l'ensemble des *événements* $E = \langle \varepsilon_0, ...\varepsilon_m\rangle$ tel que $\forall j, t_j^i \in E, t_j^f \in E$, $\varepsilon_0 = 0$ et $\forall i, \varepsilon_i \leq \varepsilon_{i+1}$.

L'état courant après un événement ε_i, noté $s[\varepsilon_i]$, est donc l'état précédent modifié par les effets de toutes les actions s'appliquant à ce moment-là (les effets et effacements étiquetés *at_end* des actions finissant en ε_i ainsi que les effets et effacements étiquetés *at_start* des actions commençant en ε_i). $s[\varepsilon_i]$ représente donc l'état du monde entre les instants ε_i et ε_{i+1}. De plus, $s[\varepsilon_0] = I$. On note $precedent(\varepsilon_{i+1}) = \varepsilon_i$.

Définition 12 : Plan valide (PDDL 2.1)

Soit $\langle F, A, I, G\rangle$ un problème de planification.
Soit $\Pi = \langle \langle t_0, \tau_0\rangle, ... \langle t_n, \tau_n\rangle\rangle$ un plan où chaque tâche τ_i correspond à l'action a_i.
Soit $E = \langle \varepsilon_0, ...\varepsilon_m\rangle$ la liste ordonnée des événements.
Un plan est valide ssi. toutes les préconditions de ses tâches sont valides. $\forall i, \forall \langle f, \phi\rangle \in Pre(a_i)$:

- Si $\phi = at\_start$: $f \in s[precedent(t_i^i)]$.
- Si $\phi = at\_end$: $f \in s[precedent(t_i^f)]$.
- Si $\phi = over\_all$: $\forall \varepsilon, t_i^i \leq \varepsilon < t_i^f, f \in s[\varepsilon]$.

De plus, si des tâches venaient à produire et consommer un fluent au même instant ou à produire et détruire un même fluent, alors ces deux tâches doivent être temporellement séparées pour lever les ambiguïtés. De plus amples détails sont disponibles dans la définition du langage [Fox et Long 2003].

Définition 13 : Plan solution (PDDL 2.1)

Soit $\langle F, A, I, G\rangle$ un problème de planification.
Soit $\Pi = \langle \langle t_0, \tau_0\rangle, ... \langle t_n, \tau_n\rangle\rangle$ un plan.
Soit $E = \langle \varepsilon_0, ...\varepsilon_m\rangle$ la liste ordonnée des événements.
p est solution du problème ssi. p est valide et $G \subset s[\varepsilon_m]$.

Exemple : PDDL 2.1

On peut étendre la modélisation en PDDL de l'exemple page 16, pour décrire un domaine temporel.
L'intégralité du domaine est présenté dans l'annexe A, page 206.
La description de l'action de déplacement d'un AAV est décrite comme suit :

```
1 (:durative-action move-aav
2   :parameters (?r - aav ?from ?to - loc)
```

```
3    :duration (= ?duration (distance-aav ?from ?to))
4    :condition (and (at start (at ?r ?from))
5                    (over all (adjacent ?from ?to))
6                    (over all (aav-allowed ?from))
7                    (over all (aav-allowed ?to)))
8    :effect (and (at end (at ?r ?to))
9                 (at start (not (at ?r ?from))) )
10 )
```

Par rapport à l'exemple de PDDL page 16, chaque fluent a été préfixé par l'instant temporel auquel il fait référence. Le robot par exemple doit initialement être au point de départ. Dès le début de l'action, le fluent définissant la position du robot est effacé pour empêcher d'autres actions nécessitant une position précise de pouvoir se déclencher. À la fin de l'action, la position du robot est définie comme étant la position finale, ce qui permet d'enchaîner avec les actions suivantes.
La durée de l'action est définie dans le problème en fonction du point de départ et du point d'arrivée. Cela permet de garder le caractère générique du domaine.
Il faut alors ajouter à la description du problème les définitions suivantes (en supposant que les déplacements d'un AGV sont plus lents que ceux d'un AAV) :

```
1 (= (distance-aav cell21 cell22) 1)
2 (= (distance-aav cell22 cell21) 1)
3 (= (distance-agv cell12 cell22) 5)
4 (= (distance-agv cell22 cell12) 5)
```

Un plan solution est alors :

```
0.001: (move-aav aav1 cell22 cell21) [1.000]
0.001: (move-agv agv1 cell22 cell12) [5.000]
1.002: (explore aav1 cell21) [1.000]
5.002: (explore agv1 cell12) [1.000]
```

Chaque ligne indique la date de début, le nom de l'action instanciée et sa durée entre crochets. Deux actions ayant la même date de début peuvent être exécutées dans n'importe quel ordre. On remarque que les actions des deux robots s'exécutent en parallèle.

Le langage PDDL2.1 étant conçu comme une extension au PDDL, toutes les descriptions PDDL antérieures sont des descriptions valides en PDDL2.1.

1.1.5 Description hiérarchique (HTN)

Le formalisme HTN (Hierarchical Task Network) [Erol, Hendler et Nau 1994] propose un cadre proche du modèle d'action de PDDL mais introduit la notion *d'action hiérarchique*, aussi appelée *action abstraite*. Les actions sont ainsi séparées en deux types : les actions élémentaires et les actions hiérarchiques. Les actions élémentaires possèdent des préconditions et des effets, comme en PDDL, et sont directement exécutables. Les actions hiérarchiques sont décrites par des préconditions et des méthodes, chaque méthode étant un moyen de réaliser cette action de haut niveau. Une méthode est un ensemble d'actions et de contraintes sur ces actions et un plan n'est valide que si toutes les actions hiérarchiques

présentes dans le plan sont instanciées à l'aide de l'une de leurs méthodes. L'objectif de la planification est alors représentée comme une action but qui doit être instanciée.

Définition 14 : Problème de planification (HTN)

Un problème de planification est défini par :

- F un ensemble de fluents ;
- A un ensemble d'actions élémentaires et d'actions abstraites ;
- $I \subseteq F$ un ensemble représentant l'état initial ;
- g une action but. $g \in A$ et g est abstraite.

Définition 15 : Action élémentaire (HTN)

Soit F un ensemble de fluents, une action est définie par :

- *name* un nom unique dans le problème considéré ;
- $Pre \subseteq F$ un ensemble de préconditions ;
- $Add \subseteq F$ un ensemble d'effets ;
- $Del \subseteq F$ un ensemble d'effacements.

Définition 16 : Action hiérarchique (HTN)

Soit F un ensemble de fluents, une action est définie par :

- *name* un nom unique dans le problème considéré ;
- $Pre \subseteq F$ un ensemble de préconditions ;
- $\mathcal{M}$ un ensemble de méthodes.

Une méthode m est un ensemble d'actions, abstraites et/ou élémentaires, et un ensemble de contraintes temporelles les ordonnant.

Si a est une action hiérarchique et m une de ses méthodes, on dit que l'on *instancie* a avec m si on ajoute dans le plan, à la place de a, l'ensemble des actions et des contraintes de m. Pour que cette instanciation soit valide, il faut que les préconditions de a soient vérifiées.

Définition 17 : Plan (HTN)

Étant donné un problème de planification HTN $\langle F, A, I, g \rangle$, un plan est défini comme un arbre de tâches de $\mathcal{T}$ dont la racine est g. Soit τ_i une tâche de cette arbre, associée à l'action a_i.

- Si a_i est élémentaire, τ_i ne peut avoir d'enfants et doit être une feuille de l'arbre.
- Si a_i est hiérarchique et τ_i est une feuille, alors on dit que τ_i n'est pas instanciée.
- Si a_i est hiérarchique et τ_i est instanciée avec la méthode $m \in \mathcal{M}(a_i)$, alors les enfants de τ_i sont exactement les tâches de m.

L'ensemble des tâches élémentaires d'un plan (avec les contraintes temporelles issues des méthodes utilisées) forment un plan. Jusqu'à la première action hiérarchique non instanciée on peut définir l'état atteint par la succession des tâches élémentaires comme en STRIPS (définition 5 page 14). Lorsqu'une action hiérarchique est présente dans le plan, il n'est pas possible d'évaluer l'état atteint après car les actions hiérarchiques ne définissent pas leurs effets. Les effets dépendent de la méthode qui sera choisie pour instancier l'action.

Définition 18 : Plan solution (représentation HTN)

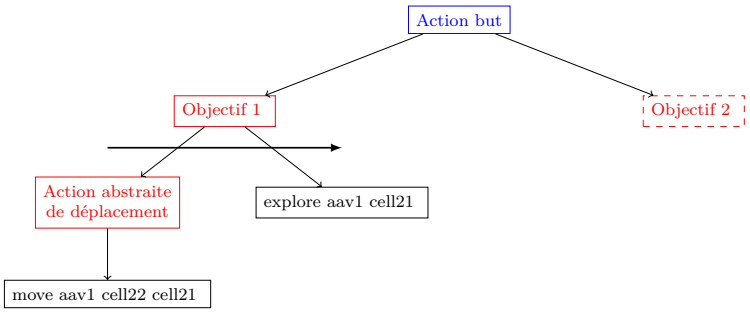
Un plan est valide ssi. toutes ses actions abstraites ont été instanciées par des méthodes valides, et que toutes les tâches élémentaires vérifient leurs préconditions : $\forall i, Pre(a_i) \subseteq s_i$ où s_i représente l'état du monde au début de la tâche τ_i.

Un plan est solution du problème $\langle F, A, I, g \rangle$ ssi. il est valide et g a été instancié.

Exemple : HTN

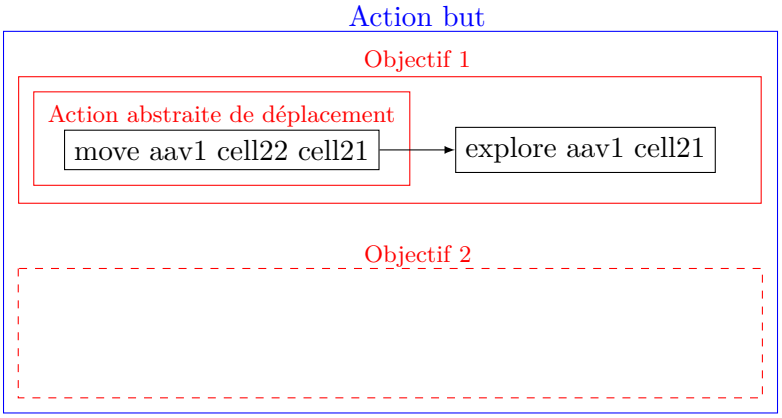
En reprenant l'exemple présenté à la figure 1.2a, on peut construire une représentation en HTN du problème.
Les actions élémentaires sont les mêmes que dans le cas d'un problème STRIPS : les actions de déplacement des robots et les actions d'observation.
L'action but contient deux actions abstraites non ordonnées : l'observation des deux points (appelés Objectif 1 et 2). Chacune de ces actions se décompose en une action abstraite de mouvement et une action élémentaire d'observation, l'action abstraite de mouvement pouvant se décomposer en éventuellement plusieurs actions élémentaires de mouvement dans le cas de problèmes plus compliqués.
Un plan partiellement instancié peut être représenté comme suit :

Figure 1.5 – Représentation d'un plan HTN.



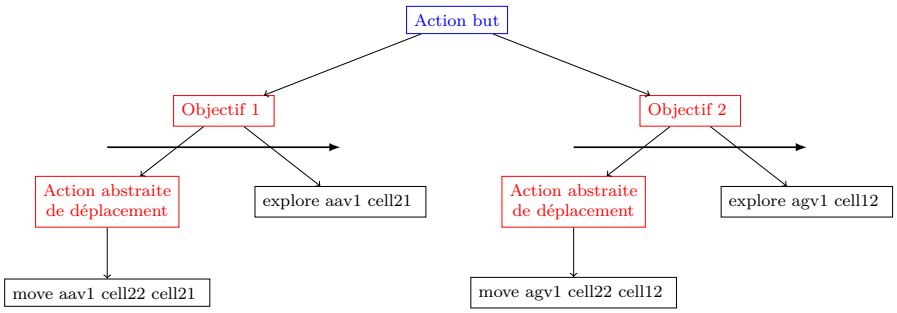
(a) Représentation sous forme d'arbre d'un plan partiellement instancié avec une action hiérarchique encore non instanciée.

L'action but est représentée en bleu et les autres actions abstraites en rouge. Les actions élémentaires sont représentées en noir. L'action abstraite non instanciée est indiquée en pointillé. Les méthodes dont les tâches sont ordonnées sont représentées par une flèche horizontale additionnelle.
Une méthode plus compacte de représentation et indiquant plus clairement les contraintes temporelles est la suivante :



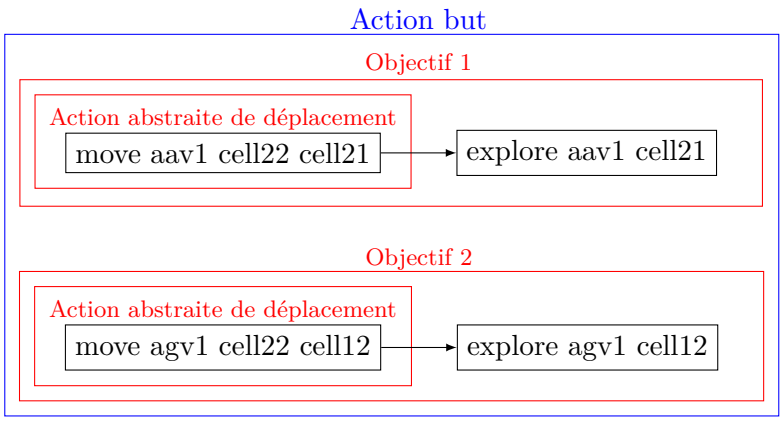
(b) Représentation sous forme compacte d'un plan partiellement instancié avec une action hiérarchique encore non instanciée.

Chaque action hiérarchique englobe ses enfants et les actions sont ordonnées de gauche à droite dans leur ordre d'exécution. La flèche noire indique une contrainte de précédence entre les actions.
Le plan solution du problème, équivalent au problème STRIPS présenté précédemment, est alors :



(c) Représentation sous forme d'arbre d'un plan complètement instancié.

Et sa représentation compacte est la suivante :



(d) Représentation sous forme compacte d'un plan complètement instancié.

Cette décomposition permet de s'adapter à différents problèmes. Par exemple si le nombre de cellules à explorer augmente, il suffit de rajouter des actions Objectifs à l'action but. Si le nombre total de cellules du problème augmente, il faut ajouter des actions de déplacement élémentaires aux actions abstraites de déplacement pour permettre des déplacements plus complexes.

La planification HTN a connu de nombreuses utilisations, en particulier car elle est adapté pour décrire des « recettes » ad-hoc, issues d'une expertise humaine, pour résoudre rapidement certains problèmes. En ce sens cette approche permet de traiter un certain nombre de problèmes qu'un planificateur générique ne peut pas résoudre au prix d'un travail plus important de la part de l'utilisateur.

1.1.6 Description par contraintes (SAT, CSP)

Une autre manière d'encoder et de résoudre un problème de planification est de le ramener à un formalisme déjà étudié et pour lequel des solveurs existent déjà. Par exemple on peut utiliser le formalisme des problèmes de satisfiabilité (appelé SAT).

Définition 19 : Problème SAT

Un problème SAT est défini comme :

- V un ensemble de variables booléennes ;
- ϕ une formule logique utilisant les opérateurs $\vee$, $\wedge$ et les variables de V.

L'objectif consiste à déterminer s'il existe un assignement des variables de V qui rend la formule ϕ vraie, et à en exhiber un. On dit alors que ϕ est satisfiable.

Théorème 1 : Equivalence STRIPS/SAT

Soit P^{STRIPS} un problème de planification STRIPS avec un horizon n. Il existe un encodage de P^{STRIPS} en un problème SAT P^{SAT} tel que P^{STRIPS} a une solution ssi P^{SAT} en a une.
De plus une solution à l'un de ces deux problèmes se transforme directement en une solution pour l'autre problème.

Preuve : Étant donné un problème STRIPS $P^{Strips} = \langle F, A, I, G\rangle$, on recherche des solutions de taille maximum n donc on peut se restreindre à ne décrire que $n+1$ états. On introduit donc une variable booléenne s_i^f pour chaque $f \in F$ et chaque $0 \leq i \leq n+1$ représentant le fait que le fluent f est vrai à l'état i. On introduit de plus une variable booléenne s_i^a pour chaque $a \in A$ et $0 \leq i \leq n$ représentant le fait que la $i^{\text{ème}}$ tâche du plan correspond à l'action a.

La contrainte ϕ est construite comme une conjonction de différentes contraintes élémentaires représentant :

- l'état initial : la valeur des s_0^f est fixée à vrai si $f \in I$, faux sinon ;
- le but : $\exists i \leq n+1, \forall f \in G, s_i^f$ est vrai ;
- l'applicabilité des tâches : $\forall a \in A, \forall f \in Pre(a), \forall i \leq n, s_i^a \implies s_i^f$;
- les effets des tâches : $\forall f \in F, \forall 1 \leq i \leq n, (s_{i-1}^a \wedge f \in Add(a) \implies s_i^f) \wedge (s_{i-1}^a \wedge f \in Del(a) \implies \neg s_i^f) \wedge (s_{i-1}^a \wedge f \notin Add(a) \wedge f \notin Del(a) \implies s_i^f = s_{i-1}^f)$.

Le choix des s_i^a est alors équivalent au choix de la $i^{\text{ème}}$ tâche du plan, et les contraintes SAT sont équivalentes aux contraintes d'un plan solution. □

L'intérêt de cette méthode est de pouvoir réutiliser des solveurs génériques existants moins spécialisés que les algorithmes de planification. Néanmoins cette méthode ne permet pas de représenter des problèmes temporels tels qu'introduit par PDDL2.1 (voir page 17).

Un autre formalisme, plus riche, peut aussi être utilisé : la programmation par contraintes (CSP). Au lieu d'utiliser des contraintes booléennes, on peut raisonner sur des variables à valeur dans des ensembles donnés, utiliser des variables continues, etc. Cela permet aussi de raisonner sur des problèmes temporels. C'est le cas de GP-CSP [Do et Kambhampati 2000]. Des techniques à base de programmation linéaire en nombres entiers ont aussi été proposées [Vossen et al. 2005].

Dans tous les cas, ce formalisme est plus proche des problèmes d'ordonnancement où les actions possibles sont connues et fixes et où la décision porte plus sur l'ordre dans lequel les réaliser et le choix du meilleur sous-ensemble d'actions à réaliser.

Une autre représentation temporelle, plus riche que PDDL 2.1, fait appel aux *timelines*. Elle se base sur une représentation en terme d'intervalles plutôt que d'états comme PDDL 2.1. Dans ce modèle, une variable peut prendre un certain nombre de valeurs (son *domaine*) et est définie comme une fonction du temps. On peut alors représenter sa valeur comme un ensemble d'intervalles distincts. C'est une représentation qui se prête bien à être utilisée à l'aide d'un gestionnaire de contraintes. C'est le cas pour EUROPA [Barreiro et al. 2012], qui utilise le langage NDDL, et IxTeT [Lemai-Chenevier 2004].

1.1.7 ANML

ANML (Action Notation Modeling Language) [D. E. Smith, Frank et Cushing 2008] est un langage qui se présente comme « de haut niveau », basé sur une représentation sous forme de timelines mais permettant de représenter aussi des décompositions hiérarchiques d'actions comme les HTN. Il permet de spécifier des effets hors du début et de la fin des actions par exemple. C'est une tentative d'unifier plusieurs langages existants avec un formalisme unique, inspiré de langages tels que IxTeT, HTN, NDDL ou SAS. De plus, un problème décrit en ANML est compilable en PDDL dans la plupart des cas.

Le premier planificateur utilisant le langage ANML a été proposé en 2014 [Dvorak et al. 2014]. A notre connaissance, aucun autre planificateur utilisant ANML n'est disponible à ce jour, ainsi qu'aucun ensemble de problèmes.

Exemple : ANML

Une action de déplacement analogue à celles présentées dans l'exemple page 19 pourrait être modélisée comme suit :

```
action Move (location from, to) {
  duration := 5 ;
  [all] { position == from :-> to ;
          batterycharge :consumes 2.0 } }
}
```

Cette action définit aussi une variation de la ressource `batterycharge`. La description d'une telle action est plus compacte et plus facilement compréhensible que celle qui utilise le PDDL.

1.1.8 Tableau récapitulatif des cadres de modélisation

Le tableau 1.1 présente une synthèse des avantages et des inconvénients des cadres présentés. La puissance de représentation décrit les types de problèmes qu'il est possible de décrire alors que la facilité d'utilisation juge plutôt la brièveté et la facilité de les décrire.

Des travaux [Nebel 2000] ont cherché à quantifier la *puissance expressive* d'un langage comme l'ensemble des problèmes qui peuvent être décrits en acceptant une augmentation linéaire ou polynomiale de la taille de la représentation, ce qui permet de comparer de manière quantitative différents langages. Les résultats synthétisés ici concernent des grandes familles de cadres et les comparaisons sont donc qualitatives. Dans chaque famille, les

spécificités de chaque langage peuvent influer sur la qualité du langage vis à vis des autres langages de la même famille.

Cadre	Puissance de représentation			Facilité d'utilisation	
	Temporel	Ressources	Hiérarchique	Écriture/Lecture	Écosystème
PDDL	Non	Oui	Non	++	+++
PDDL2.1	Oui	Oui	Non	++	+++
HTN	Non	Non	Oui	++	++
SAT	Non	Non	Non	+	++
CSP, timelines	Oui	Oui	Non	++	++
ANML	Oui	Oui	Oui	+++	+

Tableau 1.1 – Comparatif des cadres de modélisation de problèmes de planification

1.1.9 Objectifs et critères

Les modèles présentés ici ont pour la plupart la même forme d'objectif : des conditions sur l'état final atteint après l'exécution du plan sont utilisées comme contraintes dures et une métrique est utilisée comme une contrainte souple. Les contraintes dures doivent être vérifiées pour qu'une solution soit considérée comme valide, et les solutions valides sont ordonnées selon la métrique pour les départager. C'est la *planification satisfaisante* (*satisfying planning* en anglais). Les métriques couramment utilisées sont le nombre d'actions dans le plan ou la durée totale du plan pour la planification temporelle.

Une version proche est la *planification optimale* (en anglais *optimal planning*). Dans ce cas, les seules solutions valides sont celles qui minimisent la métrique. Ces problèmes sont plus difficiles à résoudre car ils imposent une contrainte supplémentaire sur la solution. Ils nécessitent plus de temps pour être résolus mais apportent une assurance de qualité sur la solution fournie. Lors des compétitions de planification des IPC, les planificateurs optimaux et satisfaisants concourent dans deux catégories différentes.

Le problème de *planification sur-contrainte* (en anglais, *over-subscription planning*) inverse les contraintes dures et les contraintes souples. La contrainte dure est une valeur maximale d'une métrique, et chaque condition sur l'état final est associée à une récompense donnée. Le but est de trouver une solution maximisant la récompense obtenue. Ce type d'objectif représente des cas où l'on essaye d'utiliser au mieux un budget donné pour accomplir un maximum de buts. Il est utilisé par exemple dans des applications spatiales, où un satellite doit utiliser au mieux son temps pour réaliser un maximum d'observations [D. E. Smith 2004].

1.2 Algorithmes de planification indépendant du domaine

Plusieurs cadres permettent de représenter un problème de planification. Intéressons-nous maintenant aux différentes méthodes de résolution qui ont été proposées. Dans la plupart des cas, les problèmes à résoudre sont au moins NP-dur [Erol, Nau et Subrahmanian 1995] et des heuristiques sont nécessaires pour guider la recherche afin de résoudre les problèmes en un temps raisonnable. Ces heuristiques peuvent être indépendantes du domaine (comme toutes celles utilisées dans les compétitions des IPC) ou adaptées au

domaine. Les heuristiques adaptées sont généralement plus performantes pour résoudre les problèmes pour lesquels elles ont été faites mais doivent être modifiées pour chaque nouveau type de problème, ce qui n'est pas toujours possible. On utilise ici une définition du domaine venant de PDDL (cf. section 1.1.3) : un algorithme est indépendant du domaine si pour fonctionner il n'a besoin que des informations contenues dans la description PDDL (actions élémentaires disponibles, état initial et état final).

La plupart des algorithmes de planification peuvent être vus comme une instance d'un algorithme abstrait. Cet algorithme (Algorithme 1) est similaire à celui présenté dans [Ghallab, Nau et Traverso 2004]. Il est très proche de la planification par raffinement (*refinement planning*) décrit dans [Kambhampati, Knoblock et Yang 1995]. Il prend comme entrée un nœud I. Chaque nœud u est un ensemble d'actions et de contraintes qui représente un ensemble de plans candidats, noté Π_u. Tous les éléments de Π_u doivent contenir les actions et les contraintes de u. L'algorithme maintient un ensemble E des nœuds non encore explorés. Différentes fonctions sont utilisées dans cet algorithme.

Algorithme 1 : Algorithme de planification générique

```
   Entrée : I, le nœud de recherche initial
 1 E = {I}
 2 tant que E ≠ ∅ faire
 3 |   Choix de u ∈ E
 4 |   E = E − {u}
 5 |   si Terminal(u) alors
 6 |   |   retourner u
 7 |   u = Raffine(u)
 8 |   B = Étend(u)
 9 |   E = E ∪ Élague(B)
10 retourner Échec
```

- *Choix* (ligne 3) : Choisit un nœud de E à explorer, en général en suivant une heuristique pour explorer en priorité les nœuds prometteurs.
- *Raffine* (ligne 7) : Ajoute de nouvelles contraintes dans u sans changer l'espace Π_u. Par exemple on peut retirer des contraintes redondantes ou au contraire expliciter une contrainte induite par d'autres déjà présentes.
- *Étend* (ligne 8) : Retourne un ensemble de nœuds B candidats à être visités ensuite. Chaque nœud v de cet ensemble est tel que $\Pi_v \subset \Pi_u$. On peut par exemple générer de nouveaux candidats en essayant l'ajout de chaque action disponible dans le plan courant.
- *Élague* (ligne 9) : Retire certains nœuds non prometteurs de l'ensemble de nœuds B. On peut par exemple retirer des nœuds qui ont déjà été visités ou pour lesquels une heuristique assure qu'ils ne mèneront pas à une solution du problème.
- *Terminal* : Retourne $Vrai$ si le nœud u permet de trouver immédiatement une solution du problème.

Une exploration des nœuds en *Last In First Out* (*i.e.* on choisit toujours le nœud le plus récent de E) correspond alors à une exploration de type *en profondeur d'abord*. Une

exploration des nœuds en *First In First Out* (*i.e.* on choisit toujours le nœud le plus ancien de E) correspond alors à une exploration de type *en largeur d'abord*.

1.2.1 Recherche dans l'espace des états

Définition 20 : Recherche en avant dans l'espace des états

En reprenant les définitions de l'algorithme 1 :

Un nœud de recherche est défini comme une séquence de tâches ordonnées, commençant à l'instant initial et où toutes les tâches présentes ont leurs préconditions vérifiées. Pour chaque nœud u, $S(u)$ est l'état atteint après l'application de cette séquence de tâches.

- Le nœud initial I est vide.
- Raffine(u) renvoie u.
- Étend(u) calcule toutes les actions applicables dans $S(u)$ et pour chacune d'entre elles crée un nouveau nœud en ajoutant une tâche correspondante à la fin de u.
- La fonction Élague retire tous les nœuds v tel que $S(v)$ a déjà été exploré, *i.e.* tel qu'on a déjà étendu un nœud menant au même état.
- La fonction Terminal(u) renvoie $Vrai$ si $S(u)$ satisfait le but.

Dans ce genre d'algorithme, la performance dépend beaucoup de la fonction de *Choix* pour orienter la recherche rapidement vers une solution. La plupart des algorithmes se basent alors sur une heuristique évaluant l'état $S(u)$.

Parmi les heuristiques utilisées, on peut citer les *landmarks*. Étant donné un problème, un *landmark* est défini comme un fait qui doit être vrai dans toutes les solutions. Plusieurs méthodes ont été proposées pour les calculer et les ordonner [Hoffmann, Porteous et Sebastia 2004 ; Richter, Helmert et Westphal 2008]. On peut alors s'en servir pour évaluer la pertinence d'un état, par exemple en évaluant le nombre de landmarks qui sont présents déjà atteint (comme le planificateur LAMA [Richter et Westphal 2010]) ou pour diriger la recherche en essayant de les résoudre rapidement. On peut citer par exemple LMBFS [Vernhes 2014] qui utilise les landmarks et leur ordre pour découper un problème de planification en sous-problèmes plus faciles.

Des variantes de cet algorithme sont possibles.

La recherche *relaxée* (en anglais *lifted*) propose de manipuler des actions partiellement instanciées, si le langage le permet, au lieu d'actions complètement instanciées. Younes et al. ont analysé les avantages et les inconvénients de ces deux représentations [Younes et Simmons 2002]. Cette méthode est souvent décrite *least commitment planning* : en instanciant les variables le plus tard possible on garde plus de flexibilité et on ne fait des choix que lorsqu'il est nécessaire de choisir. Ce principe est par exemple utilisé dans UCPOP [Penberthy et Weld 1992]

La recherche en arrière (*backward chaining*) considère plutôt la liste de tâches comme partant de la fin. Le raffinement consiste alors à ajouter des tâches au début de la liste. La difficulté dans ce cas est que l'on ne possède pas d'état courant à tout moment, ce qui

rend certaines heuristiques inutilisables.

1.2.2 Recherche à base du graphe de planification et relaxation

Dans les années 90, Blum et al. ont proposé un algorithme appelé Graphplan [Blum et Furst 1997] basé sur le calcul d'un graphe particulier, le graphe de planification. Cette technique a conduit à l'introduction de nombreuses heuristiques, dont certaines sont encore utilisées aujourd'hui. Une des hypothèses majeures consiste à ne pas prendre en compte les effacements des actions lors du calcul de ce graphe.

Le graphe de planification $\mathcal{G}$ est défini comme un graphe alternant deux types de niveaux : un niveau de fluents et un niveau d'actions. Le premier niveau est un niveau de fluents qui contient l'état initial. Chaque niveau d'actions rassemble toutes les actions applicables dans le niveau de fluents précédent. Chaque niveau de fluents suivant rassemble les fluents du niveau de fluents précédent plus ceux qui sont créés par une action du niveau précédent. Les effacements de chaque action sont ignorés. Intuitivement, le $i^{\text{ème}}$ niveau de fluents contient la liste de tous les fluents atteignables en i actions ou moins. Le graphe s'arrête une fois qu'un point fixe a été atteint.

Définition 21 : Graphe de planification

Soit $\langle F, A, I, G\rangle$ un problème de planification STRIPS.

Soit $\tilde{A} = A \cup \{\forall f \in F, noop(f)\}$ où $noop(f)$ représente une action nulle avec f pour seule précondition et pour seul effet.
Le graphe de planification $\mathcal{G}$ est défini comme $\left\langle \Lambda_f^0, \Lambda_a^0, \Lambda_f^1, \Lambda_a^1, ..., \Lambda_a^{n-1}, \Lambda_f^n \right\rangle$ où :

- $\forall i, \Lambda_f^i \subseteq F$
- $\forall i, \Lambda_a^i \subseteq \tilde{A}$
- $\Lambda_f^0 = I$
- $\forall i, \Lambda_a^i = \left\{a \in \tilde{A} \mid Pre(a) \subseteq \Lambda_f^i\right\}$
- $\forall i, \Lambda_f^{i+1} = \{f \in F \mid \exists a \in \Lambda_a^i, f \in Add(a)\}$
- n est le plus petit entier tel que $\Lambda_f^{n-1} = \Lambda_f^n$

F étant un ensemble fini, il existe un nombre fini de sous-ensembles de F. Les Λ_f^i prennent donc leurs valeurs dans un ensemble fini, ce qui prouve que n existe forcément.

L'algorithme de Graphplan consiste à dérouler le graphe de planification jusqu'à atteindre un niveau qui contient le but. On fait alors une recherche en arrière pour choisir un groupe d'actions non exclusives menant à ce but, ce qui définit un état au niveau de fluents précédent. On itère alors ce processus de choix en considérant le choix des actions de chaque niveau comme une point de backtrack possible. Une fois qu'on arrive à l'état initial, on obtient un plan solution en mettant bout à bout la liste des actions choisies à chaque niveau (sans les *noop*).

De plus, le graphe de planification peut servir à détecter itérativement les *mutex*, c'est-à-dire les groupes de fluents ou d'actions qui sont mutuellement exclusifs. On considère que deux actions sont en mutex globalement ssi. une précondition de l'une est dans la liste

des effacements de l'autre. À un niveau donné du graphe de planification, on considère que deux actions sont en mutex soit si elles sont en mutex globalement soit si deux de leurs préconditions sont en mutex au niveau de fluents précédent. À un niveau de fluents on considère que deux fluents sont en mutex si tous les couples d'actions qui les produisent sont en mutex au niveau précédent. On peut utiliser cette information pour améliorer le graphe de planification en retirant d'un niveau d'actions toutes les actions dont les préconditions sont en mutex au niveau précédent.

Lors de son introduction, Graphplan a permis des gains de performance importants vis à vis des autres planificateurs contemporains. Aujourd'hui, la recherche dans le graphe de planification est surtout utilisé comme une heuristique utilisant le problème *relaxé*.

Le problème relaxé est le problème dans lequel on ne considère pas les listes d'effacement des actions.

Définition 22 : Problème de planification relaxé

Soit $P = \langle F, A, I, G \rangle$ un problème de planification STRIPS.

On définit le problème relaxé :
$P^+ = \langle F, A^+, I, G \rangle$ où $A^+ = \{\forall \langle Pre, Add, Del \rangle \in A, \langle Pre, Add, \emptyset \rangle\}$.

Dans un tel problème, il est possible de trouver une solution (non-optimale) en un temps polynomial. Ce problème, et le graphe de planification qui lui est associé, est utilisé dans plusieurs heuristiques.

La première heuristique considérée, h^{add}, évalue un état en supposant que tous les fluents du but sont indépendants [Bonet et Geffner 2001] : en résolvant un but on ne va ni faciliter ni ralentir l'accomplissement des autres buts. On peut alors calculer le coût d'établissement d'un fluent récursivement comme étant le minimum du coût d'une action plus le coût de ses préconditions. Le coût d'une action est défini comme étant de 1 si le but est de minimiser le nombre d'actions ou comme la durée de cette action si le but est de minimiser la durée du plan. Formellement on a :

Définition 23 : h^{add} (STRIPS)

Soit $\langle F, A, I, G \rangle$ un problème de planification STRIPS.
h^{add} est défini récursivement sur un état $s \subseteq F$ et sur un fluent $f \in F$:

$$h^{add}(s) = \sum_{f \in s} h^{add}(f)$$

$$\text{avec } h^{add}(f) = \begin{cases} 0 & \text{si } s \in I \\ \min\limits_{a \in A \mid f \in Add(a)} cost(a) + h^{add}(Pre(a)) & \text{sinon} \end{cases}$$

YAHSP [V. Vidal 2011] est un exemple de planificateur qui utilise h^{add} en plus de l'utilisation de *lookahead*.

h^{ff} [Hoffmann et Nebel 2011] est une autre heuristique qui est définie sur le graphe de planification relaxé. L'idée est d'estimer la distance entre un état et le but comme la longueur d'un plan (non-optimal) résolvant le problème relaxé à partir de cet état. On peut calculer un tel plan en utilisant l'algorithme de Graphplan sur le graphe de planification relaxé. Cette heuristique ne fait pas l'hypothèse forte que les buts à atteindre

sont indépendants.

La famille des heuristiques h^m [Haslum et Geffner 2000] est basée sur les chemins critiques : on évalue la distance d'un état au but en fonction du but (ou de l'ensemble de buts) le plus difficile à atteindre. Intuitivement, la distance d'un état au but est approchée par la distance de son sous-ensemble de taille m le plus éloigné du but. Pour un ensemble E de taille inférieure à m, on cherche le minimum parmi toutes les actions du coût de l'action additionné au coût de l'état précédant cette action et aboutissant à E.

Définition 24 : h^m et h^{max}

Soit $\langle F, A, I, G\rangle$ un problème de planification STRIPS.
Soit $m \in \mathbb{N}$

h^m est défini récursivement sur un état $s \subseteq F$ et sur un ensemble critique $E \subset F, |E| = m$:

$$h^m(s) = \begin{cases} 0 & \text{si } s \subseteq I \\ \max\limits_{E \subset F, |E|=m} h^m(E) & \text{si } |s| > m \\ \min\limits_{a \in A, Del(a) \cap E = \emptyset} cost(a) + h^m((E - Add(a)) \cup Pre(a)) & \text{sinon} \end{cases}$$

Si $m = 1$, on aboutit à l'heuristique h^{max}, identique à l'heuristique h^{add} si ce n'est que la somme dans la définition 23 est remplacée par un maximum. $h^{max}(f)$ indique le premier niveau auquel on peut trouver f dans le graphe de planification relaxé.

Les h^m sont admissibles (*i.e.* elles ne sur-estiment jamais la distance d'un état au but) ce qui permet de garantir l'optimalité de la solution lors d'une recherche de type A*. Par contre, elles sont souvent moins informatives que les autres heuristiques présentées jusque-là.

1.2.3 Recherche dans l'espace des plans

Plutôt que de considérer les plans comme des séquences totalement ordonnées d'actions, la recherche dans l'espace des plans propose de les considérer comme une séquence partiellement ordonnée. Il est alors nécessaire de représenter les liens temporels entre ces actions pour garantir la cohérence finale du plan. Chaque tâche est alors associée avec un instant temporel de début et un instant temporel de fin. Toutes les contraintes sont alors stockées dans un *Simple Temporal Network* (ou STN) [Dechter, Meiri et Pearl 1991]. Cette famille est appelée POP (pour *Partial Order Planning*) ou POCL (pour *Partial Order Causal Link*) et est un exemple de recherche en arrière.

Un STN est utilisé pour gérer un ensemble de contraintes temporelles sur des instants donnés. Chaque instant peut prendre une date dans un ensemble donné (noté H). On peut utiliser $\mathbb{R}$ mais on se restreint souvent à $[0, h]$ où $h \in \mathbb{R}^+$ est l'horizon de la planification. On peut aussi utiliser $H = \mathbb{N}$ ou $[\![0, n]\!]$ pour raisonner uniquement avec des entiers. Formellement, on a :

Définition 25 : *Simple Temporal Network* (STN)

Un STN est défini par :

- H un ensemble numérique ;
- T un ensemble d'instants temporels ;
- Un ensemble de contraintes $\{t_i, t_j \in I, b \in H \mid t_i - t_j \leq b\}$.

On dit qu'un STN est cohérent s'il existe une assignation de tous les instants temporels dans H telle que toutes les contraintes temporelles sont respectées. Pour représenter une contrainte de précédence (*i.e.* t_i doit arriver avant t_j, noté $t_i \prec t_j$), on utilise des contraintes de la forme $\{t_i, t_j, \epsilon\}$ avec ϵ petit (pour une précédence stricte) ou 0 (si les deux instants peuvent être simultanés).

De plus, il est nécessaire de conserver dans le plan les *liens causaux*. Défini entre deux tâches, un lien causal représente le fait que la première tâche produit un effet nécessaire à la seconde tâche dans le plan. Il induit naturellement une précédence $t_s \prec t_e$ où t_s est l'instant temporel de création de f par la première tâche et t_e est l'instant temporel nécessitant f pour la seconde tâche.

Définition 26 : Lien causal (POP)

Un lien causal est défini par sa tâche de début et son instant de début (τ_s, t_s), sa tâche de fin et son instant de fin (τ_e, t_e) et un fluent f avec $f \in Add(\tau_s)$ et $f \in Pre(\tau_e)$.

Définition 27 : Plan partiel (POP)

Soit $\langle F, A, I, G \rangle$ un problème de planification STRIPS.
Un plan est défini par :

- Un ensemble de tâches $\{\tau_i = \langle a_i, t_i^s, t_i^e \rangle\}$ où a_i est une action de A et où les t_i^s, t_i^e représentent des instants temporels ;
- Un ensemble de liens causaux CL ;
- Un ensemble de liens temporels TL, chaque lien $\langle t_s, t_e \rangle$ indiquant une précédence $t_s \prec t_e$.

De plus, chaque plan contient 2 tâches particulières : une correspondant aux conditions initiales (précédent toutes les actions et avec $Add = I$) et une correspondant aux buts (postérieure à toutes les actions et avec $Pre = G$).

Le point temporel t_i^s représente l'instant de début de la tâche τ_i et t_i^e représente sa fin au sein d'un STN associé au plan. L'ensemble des points temporels est partiellement ordonné par les éléments du plan : les liens causaux et les liens temporels induisent des contraintes de précédence entre ces points. La durée de la tâche τ_i est représentée par les contraintes dans le STN $\left\{t_i^s, t_j^e, dur\right\}$ et $\left\{t_i^e, t_j^s, -dur\right\}$.

Étant donné un plan, il est possible de calculer la liste des « problèmes » qui empêchent

son exécution, appelé *défauts*.

Définition 28 : Défaut (POP)

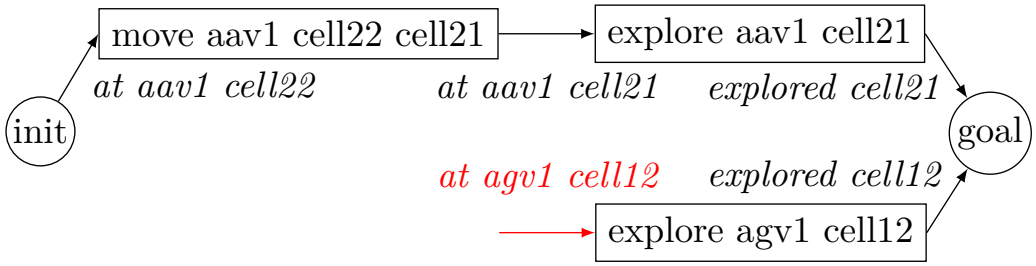
Étant donné un plan POP, on définit deux types de défauts :

- Lien ouvert : il n'existe pas de lien causal pour une précondition d'une tâche du plan ;
- Menace : une tâche effaçant le fluent d'un lien causal peut arriver pendant ce lien causal.

Exemple : Plan POP

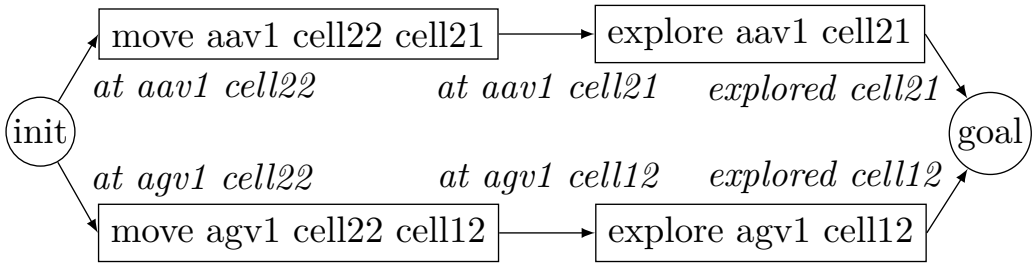
En reprenant l'exemple présenté page 19 ainsi que la description des actions, on présente ici un plan partiel (figure 1.6a) et un plan solution (figure 1.6b).
Pour alléger la figure, on suppose que la seule précondition des actions de mouvement est `at ?r ?from`.

Figure 1.6 – Exemple de plans POP.



(a) Exemple d'un plan partiel POP présentant un défaut.

Les flèches représentent les liens causaux et les deux actions particulières de début et de fin sont représentées par des cercles. Les fluents associés à chaque lien causal sont indiqués en italique à coté. La figure précédente possède une action dont une des préconditions n'est pas garantie par le plan courant : l'action d'observation nécessite la présence de l'AGV au lieu d'observation. Ce lien ouvert est représenté en rouge. Pour résoudre ce défaut, il est nécessaire d'ajouter un lien causal pour ce fluent. C'est le cas dans le plan suivant, qui présente un plan solution du problème.



(b) Exemple d'un plan POP solution du problème d'exemple.

Avec cette représentation, on peut définir une procédure de recherche adaptée de

l'Algorithme 1 page 28.

Définition 29 : Recherche dans l'espace des plans

Soit $\langle F, A, I, G\rangle$ un problème de planification STRIPS. En reprenant les définitions de l'algorithme 1 :

Un nœud de recherche est défini comme un plan partiel (définition 27).

- Le nœud initial I est un plan « vide » : il contient uniquement les deux tâches obligatoires de début et de fin.
- Raffine(u) renvoie u.
- Étend(u) choisit un défaut de ce plan et renvoie une liste de plans résolvant ce défaut. On résout un lien ouvert en insérant une nouvelle tâche et un nouveau lien causal. On résout une menace en introduisant une contrainte temporelle qui contraint l'action menaçante à être avant ou après le lien causal menacé.
- La fonction Élague retire tous les nœuds v tel que le STN de v n'est plus cohérent.
- La fonction Terminal(u) renvoie $Vrai$ si u ne possède pas de défaut.

Un plan partiel sans défaut définit une famille de solutions au problème. Toute instanciation valide des instants temporels du STN produira un plan solution du problème. En effet, chaque précondition de chaque tâche sera assurée par un lien causal (donc elle sera créée par une action et ne sera pas effacée entre-temps). De plus, le but sera atteint car les liens causaux reliés à l'action finale du plan le garantissent.

Cet algorithme nécessite deux heuristiques : l'heuristique de sélection du plan à étendre et l'heuristique de sélection du prochain défaut à résoudre (dans le fonction Étend de l'algorithme 1). Plusieurs implémentations ont été proposées comme UCPOP [Penberthy et Weld 1992], VHPOP [Younes et Simmons 2003] et RePOP [Nguyen et Kambhampati 2001]. VHPOP (*Versatile Heuristic Partial Order Planner*) propose une comparaison de plusieurs heuristiques applicables.

Concernant l'heuristique de choix des plans, une des différences majeures avec la recherche dans l'espace des états est que, étant donné un plan, on ne peut pas calculer d'état courant. Toutes les heuristiques basées sur l'évaluation d'un état sont donc inutilisables. Les heuristiques h^{add} (définition 23 page 31) et h^{max} (définition 24 page 32) utilisent l'estimation du coût de chaque fluent du but étant donné un état de départ. Elles sont donc transposables dans ce cas, et permettent d'évaluer le coût d'établissement d'un lien causal. Avec le même principe, on peut définir h^{add} et h^{max} comme respectivement la somme et le maximum des coûts des différents liens ouverts du plan. Des évolution sont proposées, comme h^{add}_{reuse} qui prend en compte les fluents déjà créés mais pas encore liés par un lien causal ou l'utilisation de l'*effort* (ici défini comme étant le nombre de raffinements à réaliser pour atteindre le but) nécessaire pour départager les égalités.

Concernant l'heuristique de choix des défauts, les défauts peuvent être classés en fonction de leur type (menace ou lien ouvert), du nombre de plans résolvant ce défaut, de sa place dans la liste des défauts (pour résoudre en priorité les plus récents/plus anciens),

etc. VHPOP [Younes et Simmons 2003] propose une comparaison des performances pour plusieurs de ces stratégies combinées. Il montre entre autres que choisir en priorité les menaces puis les liens ouverts issus de l'introduction de la dernière action ajoutée est profitable. Cela permet au planificateur de se concentrer sur un sous-but donné tout en permettant de choisir le meilleur défaut à résoudre parmi tous ceux qui participent à établir ce sous-but.

La représentation POP a l'avantage de garder des traces du raisonnement dans le plan : avec la structure des liens causaux on peut savoir pourquoi une action a été introduite dans le plan et pourquoi les contraintes ont été introduites. Cela permet d'« expliquer » les plans à un humain [Bercher et al. 2014] en cas d'interactions avec un opérateur. Elle permet aussi de faciliter la fusion de plans calculés indépendamment [Hashmi et Seghrouchni 2010] en un seul plan sans conflit.

1.3 Algorithmes de planification dépendant du domaine

Certains problèmes de planification restent encore trop complexes pour être résolus dans un temps raisonnable par les approches présentées précédemment. Ainsi, des planificateurs pouvant être adaptés aux problèmes à résoudre ont vu le jour. L'idée est de concevoir et d'utiliser un formalisme qui permette à l'humain d'écrire des règles qui vont guider la recherche plus efficacement que les heuristiques générales.

1.3.1 Recherche hiérarchique

Le formalisme de ce type le plus utilisé est celui des *Hierarchical Task Network* (HTN), décrit section 1.1.5. L'idée est non pas d'ajouter des actions pour satisfaire les buts, mais d'utiliser des méthodes décrites par l'utilisateur. Ce formalisme a été décrit initialement pour le planificateur SHOP [Nau, Cao et al. 1999].

Dans le cas d'une recherche totalement ordonnée (*i.e.* toutes les méthodes sont décrites comme une séquence totalement ordonnée d'actions), on peut définir un algorithme simple de recherche en avant.

Définition 30 : Recherche HTN totalement ordonnée

Soit $\langle F, A, I, G\rangle$ un problème de planification HTN. En reprenant les définitions de l'algorithme 1 :

Un nœud de recherche est défini comme un ensemble de tâches.

- Le nœud initial est un ensemble de tâches comprenant uniquement une tâche associée à l'action G.
- Raffine(u) renvoie u.
- Étend(u) étend la première tâche abstraite non instanciée du plan (toutes les tâches étant complètement ordonnées, cette tâche est unique). Pour chaque méthode dont les préconditions sont vérifiées, Étend(u) renvoie le plan associé à l'instanciation de cette tâche
- La fonction Élague retire tous les nœuds v tel qu'une action élémentaire n'a

pas ses préconditions valides si toutes les tâches abstraites la précédant sont instanciées.

– La fonction Terminal(u) renvoie $Vrai$ si u ne possède plus de tâches abstraites non instanciées.

Certaines des contraintes imposées dans la définition 30 peuvent être levées, comme l'a fait SHOP2 [Nau, Au et al. 2003].

On peut par exemple ne pas se restreindre à des tâches totalement ordonnées. Cela nécessite de faire des choix sur l'ordre de résolution des tâches en autorisant le *backtrack*.

SHOP2 est un planificateur *relaxé* (comme défini à la section 1.2.1) : le raisonnement se fait sur des tâches comprenant des variables. Étend(u) doit alors aussi se préoccuper des tâches élémentaires et doit renvoyer une liste de leurs instanciations possibles.

Une autre modification consiste à autoriser le planificateur à faire de l'ajout opportuniste d'actions pour résoudre un but. Par exemple s'il n'existe pas de solution d'un problème se conformant à la description hiérarchique donnée, cela ne veut pas dire qu'il n'existe pas de solution au problème. Cela permet au planificateur d'ajouter des tâches en dehors de l'instanciation d'une tâche déjà présente dans le plan. La hiérarchie des actions est donc un ensemble d'arbres plutôt qu'un arbre unique.

Certains travaux proposent d'associer des effets aussi aux actions abstraites. L'idée n'est pas de décrire exactement les effets de l'action (car ils vont dépendre des méthodes choisies) mais de décrire les effets escomptés pour permettre à la recherche de s'orienter plus rapidement vers les méthodes adaptées. C'est le cas de la *planification angélique* [Marthi, Russell et Wolfe 2007] qui étiquette chaque action automatiquement, avec l'ensemble de fluents créé dans tous les cas et dans au moins un cas. Cela permet de déterminer les effets potentiels et garantis de l'action et de guider la recherche.

1.3.2 Recherche hybride

Une technique de planification rassemblant la recherche hiérarchique et la recherche dans l'espace des plans a été proposée, sous le nom de planification hybride. L'avantage est de permettre l'ajout opportuniste d'actions (ce qui permet de se contenter d'une description hiérarchique non exhaustive) tout en ayant les gains en performance associés avec la planification HTN. Cela permet aussi de raisonner sur des problèmes temporels (avec des actions concurrentes par exemple), ce que ne permet pas la planification HTN. Par rapport à la planification angélique, les effets des actions hiérarchiques viennent avec la définition des méthodes et ne décrivent que les fluents qui peuvent être créés à coup sur.

O-Plan [Currie et Tate 1991] et DPOCL [Young, Pollack et Moore 1994] sont parmi les premiers planificateurs apparus qui utilisent ce principe. D'autres algorithmes ont été proposés depuis tel que celui proposé par Kambhampati [Kambhampati, Mali et Srivastava 1998] ou FAPE [Dvorak et al. 2014]. Schattenberg propose dans son mémoire de thèse un algorithme nommé PANDA et un cadre permettant d'unifier les techniques POP et HTN ainsi que des techniques d'ordonnancement [Schattenberg 2009].

L'idée principale est de reprendre la recherche dans l'espace des plans (telle que définie page 35) en ajoutant un nouveau type de défaut : la présence d'une action abstraite non instanciée. La méthode de résolution de ce défaut est alors de tester toutes les méthodes

possibles pour l'instancier. Cela induit un changement minimal dans l'algorithme de recherche, mais les heuristiques doivent être adaptées pour gérer ce nouveau défaut.

[Estlin, Chien et Wang 1997] présente le travail qui a été effectué à la NASA sur deux planificateurs hybrides et les leçons qu'ils en ont tirées. Plus particulièrement, ils défendent une séparation importante entre les parties POP et HTN de la représentation.

SIADEX [Castillo et al. 2006] est un système qui a été déployé pour aider à la planification d'opérations de lutte contre les feux de forêts, basé sur un raisonnement temporel incorporé à une recherche HTN. Il n'autorise pas l'ajout opportuniste d'action, mais raisonne sur un STN associé à un plan hiérarchique, ce qui le rapproche de la planification hybride.

1.3.3 Autres formalismes de la connaissance experte

D'autres formalismes de modélisation de la connaissance experte ont été proposés.

On peut citer par exemple TALPlan [Kvarnström et Doherty 2000] et TLPlan [Bacchus et Kabanza 2000] qui ont proposé tous les deux un formalisme permettant d'utiliser la logique temporelle pour élaguer rapidement des branches de l'arbre de recherche. Plus précisément, les règles permettent de préciser l'évolution temporelle des plans. On peut par exemple y décrire que si un but donné est atteint, alors il ne doit jamais être effacé. Ou à partir du moment où une propriété est vraie, alors elle doit rester vraie dans tous les états suivants. Utilisé dans le cadre d'une recherche en avant dans l'espace des états (cf. définition 20 page 29), cela permet d'améliorer la fonction Élague : elle retire tous les plans qui ne vérifient pas ces contraintes.

Conclusion

Dans un système robotique, les actions à réaliser sont calculées en fonction des perceptions du robot et de ses objectifs. L'algorithme qui décide de ces actions est appelé planificateur : son rôle est de calculer un plan, *i.e.* un ensemble d'actions et de contraintes temporelles entre ces actions.

Plusieurs cadres existent pour représenter ces problèmes, regroupés à la section 1.1. Certains permettent de représenter des problèmes plus ou moins riches (par exemple en considérant des actions avec une durée et une synchronisation entre ces actions) et avec plus ou moins de facilité pour l'opérateur à écrire ces problèmes.

En plus de la description, plusieurs types d'algorithmes de résolution de ces problèmes existent. Certains sont indépendant du problème à traiter (section 1.2). Cela veut dire qu'il est possible de substituer facilement un algorithme à un autre (s'ils utilisent le même cadre de modélisation), mais aussi que les méthodes de recherche doivent être génériques. Certains algorithmes sont plus ou moins rapides, adaptés à certains domaines ou permettent de résoudre des problèmes plus riches. Une autre grande famille d'algorithme sont les algorithmes dépendant du domaine à traiter (section 1.3). Ils nécessitent chacun de l'information supplémentaire, souvent adapté à l'algorithme, mais permettent des gains de performance importants. Cette famille est la plus utilisée en pratique.

2

Réparation

Sommaire

Comme l'illustre la figure 1.1 page 7, un système robotique évoluant dans un monde réel doit régulièrement mettre à jour son plan. Cela peut être dû à de nouvelles perceptions qui viennent changer ce que le robot sait de son environnement, un changement de ses buts ou un effet imprévu d'une action (par exemple une action qui échoue à accomplir ses effets attendus). Le plan initial est calculé selon les algorithmes présentés dans le chapitre précédent et ce plan doit être mis à jour pendant son exécution.

Dans ces cas, il est possible de recalculer un plan de la même manière que lors du calcul du plan initial, sans tenir compte du plan courant, mais cela présente plusieurs désavantages :

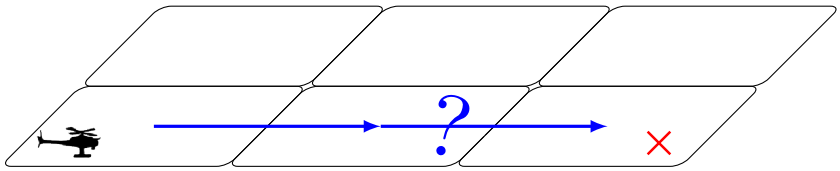
- Le nouveau plan peut être extrêmement différent du premier plan, ce qui induit un comportement contre-intuitif où le plan change pour une raison incompréhensible pour l'opérateur humain supervisant le robot.
- Calculer un nouveau plan à partir de zéro peut être beaucoup plus coûteux en temps de calcul que de modifier le plan actuel.
- Il n'est pas possible de continuer l'exécution en partie du plan courant en parallèle du calcul car il n'y a aucune garantie que les prochaines actions à exécuter seront dans le nouveau plan.

Nous nous intéressons dans ce chapitre à toutes les méthodes de calcul d'un nouveau plan partant d'un plan déjà existant. Nous appellerons ce processus *réparation* par opposition à la *replanification* qui consiste à ne pas tenir compte du plan actuel.

Exemple : Réparation

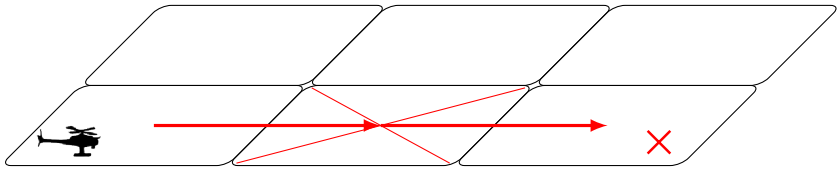
En reprenant l'exemple du chapitre précédent, on peut définir un problème de réparation relativement simple. Supposons qu'il n'y ait qu'un seul véhicule et qu'un seul point à explorer. Le chemin le plus court passe par un lieu dont la traversabilité n'est pas garantie.
Le problème est représenté figure 2.1a. Le chemin en bleu correspond au plan calculé initialement.

Figure 2.1 – Exemple de problème de réparation.



(a) Problème initial.

Une fois le plan calculé, son exécution est lancée. Et dès le premier mouvement, l'action de déplacement échoue à cause d'un obstacle imprévu qui rend le chemin impraticable. Cette situation est illustrée figure 2.1b. Les actions devenues impossibles sont indiquées en rouge.



(b) Problème à réparer : le plan actuel n'est plus faisable.

On cherche alors un plan pour résoudre ce nouveau problème, avec une case bloquée.

2.1 Intérêt de la réparation, critères d'évaluation et définition du problème

2.1.1 Intérêt de la réparation et critères associées

La première question qui se pose est l'intérêt de la réparation. Si on possède déjà des algorithmes de planification, pourquoi en chercher d'autres spécialement adaptés à la réparation alors qu'il suffirait de replanifier ?

Fox et al. proposent d'utiliser la *stabilité* d'un plan pour mesurer la quantité de changements entre le plan initial et le plan réparé.

Définition 31 : Stabilité d'un plan

Soit Π_1 le plan initial et Π_2 le plan réparé.

La différence entre ces deux plans est le nombre d'actions de Π_1 qui ne sont pas dans Π_2 plus le nombre d'actions de Π_2 qui ne sont pas dans Π_1.
La stabilité est l'opposée de la différence entre Π_1 et Π_2.

Ils argumentent que cette stabilité est une propriété importante pour plusieurs raisons.

- Elle permet de réduire la charge cognitive d'un opérateur humain supervisant le système robotique. En effet, tout changement brutal du plan exécuté nécessite un effort important de la part de l'opérateur pour comprendre et suivre l'exécution d'un plan. Elle permet aussi de faciliter la synchronisation entre agents autonomes en permettant aux agents de garder au mieux leurs engagements entre deux réparations.

- Elle permet de mieux prédire les suites d'actions qui seront exécutées, ce qui permet de faciliter les transitions entre ces actions. [Stulp et Beetz 2005] par exemple montre comment apprendre et modifier l'exécution d'actions élémentaires pour assurer un meilleur comportement global.

- La réparation permet de trouver une solution plus rapidement que la replanification dans la plupart des cas, tout en gardant des plans de bonne qualité. Par exemple conserver la stabilité permet de bénéficier de certains raisonnements qui ont été faits pendant la recherche du plan initial.

Ces deux derniers points sont démontrés de manière empirique sur un ensemble de problèmes différents [Fox, Gerevini et al. 2006]. D'autres études ont aussi montré le gain de performance de la réparation [Chien et al. 2000 ; Krogt et Weerdt 2005].

Une autre application proche de la réparation est l'*adaptation de plan* [Cox, Muñoz-Avila et Bergmann 2005 ; Lee-Urban 2012]. L'idée n'est pas de réparer un plan en cours d'exécution, mais de trouver un plan solution à un problème étant donné certaines solutions d'exemple. Par exemple, étant donné plusieurs plans types d'évacuation d'un bâtiment, le jour où un sinistre arrive, un plan d'évacuation adapté au cas en cours doit être calculé. Même si la situation actuelle ne correspond pas complètement à une des situations correspondant à un des plans types, ces algorithme les utilisent pour trouver un plan adaptée à la situation actuelle. Ces deux approches, la réparation et l'adaptation, consistent à utiliser un (ou plusieurs) plan(s) proche(s) de la solution pour trouver plus rapidement une bonne solution. Certains algorithmes cherchent par exemple le plan connu

le plus proche de la solution pour l'adapter, ce qui revient ensuite à ce que l'on a appelé *réparation*. C'est le cas de CAPlan/CBC [Muñoz-Avila et Weberskirch 1996].

En général, il est nécessaire de chercher un compromis entre l'optimisation de la qualité du plan (mesurée en fonction du critère du problème) et celui de la stabilité de la solution. [Cushing et Kambhampati 2005] argumente que la qualité d'un plan (en cours d'exécution) doit se mesurer en prenant en compte les engagements que l'agent a déjà produit. Ainsi il propose d'intégrer les engagements (rendez-vous, interactions, etc. avec les autres robots) qui sont encore satisfaits dans le plan réparé dans la métrique utilisée pour juger de la qualité d'un plan.

2.1.2 Définition de la réparation

Une première approche consiste à changer les conditions initiales et les buts à atteindre sans changer les fluents existants ou les actions disponibles.

Après l'exécution d'une action, il est possible de retirer cette action du plan et de changer les conditions initiales pour représenter l'état obtenu après l'application de cette action. Une action qui échoue ou qui ne produit pas ses effets escomptés peut être modélisée avec ce formalisme.

C'est le cas de GPG [Gerevini et Serina 2000], qui propose une évaluation sur un benchmark composé de problèmes générés en changeant uniquement les conditions initiales ou finales. RepairSHOP [Warfield et al. 2007] utilise aussi le même principe pour générer ses benchmarks : seul l'état initial est changé.

Cette approche est aussi utilisée dans les cas d'*adaptation de plans*, où les plans donnés ont été calculés dans certaines situations et où l'on cherche à trouver une solution à une autre situation donnée.

Néanmoins, cette approche ne permet pas de prendre en compte d'autres événements qui peuvent survenir [Cushing et Kambhampati 2005]. Par exemple de nouvelles actions disponibles et/ou impossibles, un changement des effets d'une action, une nouvelle méthode disponible, etc. Il est possible de définir aussi un processus de réparation où le plan initial a été calculé pour résoudre un problème différent (avec des actions différentes par exemple). Néanmoins, plus les actions disponibles changent, plus le raisonnement qui a mené au plan initial est obsolète et ne peut pas être réutilisé. C'est le cas dans [Koenig, Furcy et Bauer 2002] par exemple, qui évalue la réparation en interdisant une action utile du plan pour modifier le graphe qui exploré pendant la recherche : on réagit alors à une action utilisée qui devient impossible.

2.2 Réparation d'un plan

Plusieurs algorithmes permettent d'utiliser (ou nécessitent dans certains cas) un plan proche de la solution.

2.2.1 Recherche locale

[Krogt et Weerdt 2005] propose un algorithme adapté de la planification par raffinement qui a été proposé comme algorithme général pour la planification (Algorithme 1 page 28). L'idée est de ne pas se limiter à étendre les nœuds en ajoutant des contraintes pour accomplir un but. On s'autorise aussi à « revenir en arrière » en retirant des contraintes du nœud actuel. L'Algorithme 2 obtenu utilise les mêmes notations que l'Algorithme 1, avec deux nouvelles fonctions :

– *ÉtendEnAvant* (ligne 11) : Retourne un ensemble de nœuds B candidats à être visités ensuite. Chaque nœud v de cet ensemble est tel que $\Pi_v \subset \Pi_u$. On peut par exemple générer de nouveaux candidats en essayant l'ajout de chaque action disponible dans le plan courant.

– *ÉtendEnArrière* (ligne 9) : Retourne un ensemble de nœuds B candidats à être visités ensuite. Chaque nœud v de cet ensemble est tel que $\Pi_u \subset \Pi_v$. On peut par exemple retirer une action du plan si elle empêche un but d'être atteint ou si elle est inutile.

Algorithme 2 : Algorithme de réparation générique

```
   Entrée : I, le nœud de recherche initial
 1 E = {I}
 2 tant que E ≠ ∅ faire
 3 |   Choix de u ∈ C
 4 |   C = C − {u}
 5 |   si Terminal(u) alors
 6 |   |   retourner u
 7 |   u = Raffine(u)
 8 |   si Choix de retirer des contraintes alors
 9 |   |   B = ÉtendEnArrière(u)
10 |   sinon
11 |   |   B = ÉtendEnAvant(u)
12 |   E = E ∪ Élague(B)
13 retourner Échec
```

Pour garantir la terminaison de l'algorithme, comme cet algorithme permet d'ajouter et de retirer des éléments au plan, il est nécessaire de mettre en place des stratégies pour éviter les boucles infinies (par exemple un cycle infini qui consisterait à ajouter puis retirer une même action à l'infini). Une technique consiste à garder un historique des nœuds visités pour élaguer tous les nœuds déjà explorés (où déjà dans E).

Cette algorithme décrit un processus de recherche locale dont l'objectif est de modifier itérativement un plan. Dans le cas d'une planification, le point de départ de la recherche peut être le plan vide. Dans le cas d'une réparation, le point de départ est le plan initialement calculé. L'idée est de mélanger les étapes de retraits de contraintes et d'ajouts de contraintes. Cela permet de revenir sur un choix qui a été fait précédemment tout en gardant le reste du plan inchangé.

GPG [Gerevini et Serina 2000] s'autorise par exemple à ajouter et à retirer des actions pour réaliser une recherche locale. Il s'appuie sur une planification dans le graphe de planification (cf. section 1.2.2). Il restreint aussi l'intervalle temporel sur lequel il s'autorise à agir pour limiter la recherche en cherchant les incohérences du plan actuel et en se limitant à une portion les contenant. Pour garantir la complétude de la recherche, ces bornes sont itérativement élargies si aucune solution n'est trouvée.

Cette approche est particulièrement utile pour améliorer des plans déjà existants : le plan fourni par un planificateur est alors utilisé comme point de départ.

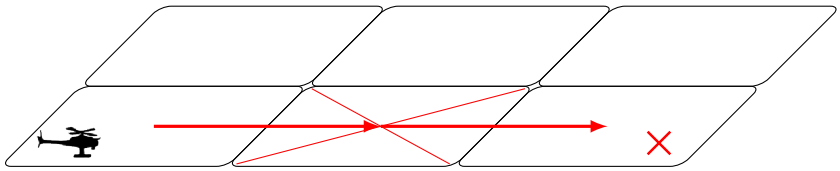
C'est le cas des travaux présenté dans [Nakhost et Müller 2010] qui proposent de calculer automatiquement les actions inutiles d'un plan et de faire une recherche locale pour trouver de meilleures solutions dans un cadre de planification classique (*i.e.* non

temporelle). [Bajada, Fox et Long 2014] propose aussi une méthode de recherche locale, mais dans le cadre de la planification temporelle en cherchant à optimiser une fonction de coût éventuellement différente de la longueur temporelle du plan.

Exemple : Réparation par recherche locale

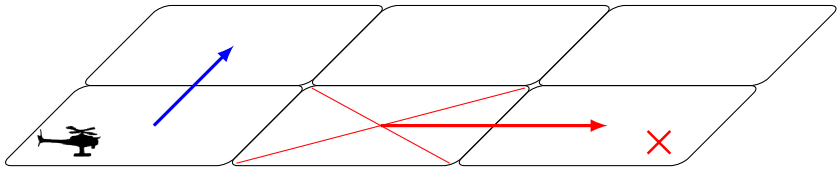
En repartant du problème de réparation présenté page 40 :

Figure 2.2 – Exemple de réparation par recherche locale.



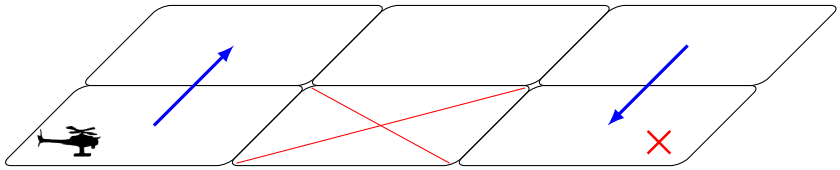
(a) Problème à réparer : le plan actuel n'est plus faisable.

Le premier changement local peut consister à changer l'action de déplacement partant de la position initiale. Il y a une seule possibilité d'action valide, ce qui donne alors :



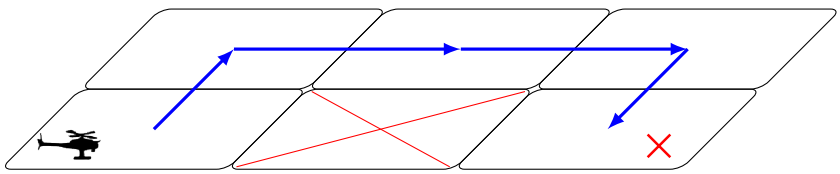
(b) Problème en cours de réparation.

De même, l'action finale n'est pas utile car elle n'est pas applicable : il n'est pas possible de partir d'un lieu inaccessible. Dans ce cas simple, il n'existe qu'une seule autre alternative. On obtiens alors :



(c) Problème en cours de réparation.

A partir de ce plan, les heuristiques de recherche peuvent guider la recherche vers l'ajout des actions de déplacement manquantes. Ce qui aboutit alors au plan suivant, prêt à être exécuté :



(d) Problème réparé.

2.2.2 Dé-raffinement puis raffinement

Une des principales difficultés de la recherche locale pour la réparation est de savoir quelles sont les contraintes à retirer pour garantir que l'on se rapproche d'une solution (ou d'une meilleure solution) sans refaire en permanence les mêmes choix.

Pour simplifier ce problème et se rapprocher des algorithmes de planification déjà existants, il est possible de séparer la réparation en deux grandes étapes : on retire d'abord les actions et les contraintes qui empêchent d'atteindre le but avant d'en ajouter (cette dernière étape étant très proche de la planification). Dans l'Algorithme 2, cela consiste à faire tous les retraits de contraintes au début de la recherche puis à effectuer seulement des ajouts. Cela permet d'éviter les boucles infinies sans avoir à garder d'historique. De plus, une fois que tous les retraits ont été effectués, on peut utiliser des algorithmes classiques présentés dans le chapitre précédent.

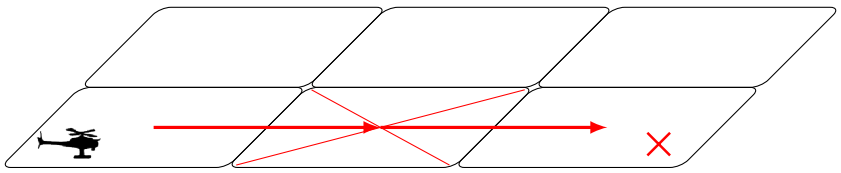
POPR [Krogt et Weerdt 2005] est un algorithme tiré de VHPOP qui utilise ce principe. Il propose de plus d'utiliser les mêmes heuristiques que celles servant à la planification pour calculer quelles actions retirer du plan dans la première étape. L'idée d'utiliser une recherche dans l'espace des états (cf. section 1.2.3) est de remarquer que le retrait d'actions d'un plan partiel produit toujours un plan partiel. Cela permet de juger la pertinence d'un plan partiel en utilisant les mêmes heuristiques que celles qui sont utilisées pour guider la planification. Les actions à retirer sont calculées en utilisant les liens causaux du plan pour déterminer des sous-ensembles d'actions dépendantes les unes des autres pour toutes les retirer d'un coup. Les heuristiques jugeant les plans partiels sont alors utilisées pour déterminer quels sont les meilleurs ensembles d'actions à retirer. Cet algorithme a un fonctionnement itératif : on essaye d'abord de retirer toutes les actions qui sont à un lien causal de l'état initial ou du but. Si aucune solution n'existe, on cherche parmi les actions étant à 2 liens causaux, etc. L'algorithme s'arrête quand on a initialement retiré toutes les actions du plan, et que la réparation (qui est dans ce cas une replanification) a échoué.

Replan [Boella et Damiano 2002] propose une méthode similaire mais basé sur les HTN (cf. section 1.3.1) pour déterminer quelles actions doivent être retirées avant de relancer une planification. L'idée est de chercher la première tâche qui n'est plus applicable et de la retirer en désinstanciant son parent dans l'arbre. Si aucune autre instanciation n'est possible, alors on continue en désinstanciant l'ancêtre suivant, et ainsi de suite jusqu'à trouver une solution ou à désinstancier l'action but (ce qui revient à replanifier).

Exemple : Réparation par dé-raffinement puis re-raffinement

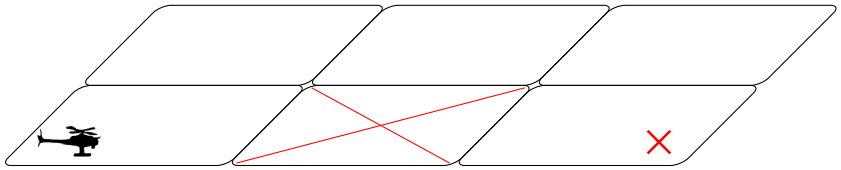
En repartant du problème de réparation présenté page 40 :

Figure 2.3 – Exemple de réparation par dé-raffinement puis re-raffinement.



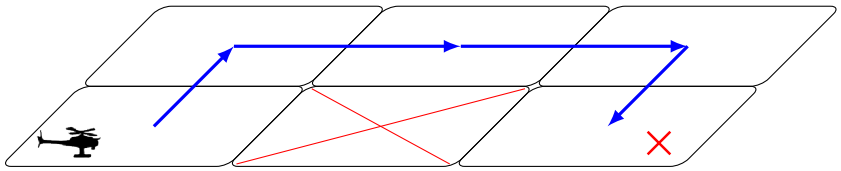
(a) Problème à réparer : le plan actuel n'est plus faisable.

Dans ce cas de figure, la partie de dé-raffinement est simple : toutes les actions encore présentes dans le plan sont invalides et doivent donc être retirées. On aboutit alors à la situation suivante.



(b) Problème en cours de réparation : certaines actions ont été retirées.

A partir de cette situation, une planification peut avoir lieu. Dans ce cas de figure, on replanifie complètement : toutes les actions possibles ont été retirées du plan. Ce n'est pas forcément le cas, par exemple si d'autres robots avaient des actions prévues, elles ne seraient pas forcément impactées par cette réparation. On aboutit alors à la même solution que dans le cas d'une recherche locale :



(c) Problème réparé.

2.2.3 Utilisation de règles spécifiques

Tout comme certains systèmes de planification utilisent des règles spécifiques (comme la planification hiérarchique), certains systèmes de réparation utilisent des règles explicites de réparation.

C'est le cas de O-Plan [Drabble, Dalton et Tate 1997]. Tous les problèmes pouvant survenir pendant l'exécution sont répertoriés et une ou plusieurs méthodes de résolution sont écrites. Ainsi, quand un problème survient, il est identifié et une des méthodes de réparation adaptées est essayée. Par contre les méthodes proposées ne contiennent que des actions à ajouter, pas de retrait d'actions. De plus, cet algorithme ne peut que gérer des événements qui ont été potentiellement prévus, pas les événements imprévus pour lesquels aucune méthode n'a été fournie.

Des modifications [Wang et Chien 1997] à O-Plan ont été proposées pour se passer de telles méthodes si l'on suppose que les éléments du problème ont un état par défaut atteignable depuis tout état. La réparation consiste alors à remettre dans leur état par défaut les éléments nécessaires pour atteindre le but depuis ces états par défaut.

CHEF [Hammond 1990] est un autre exemple de planificateur qui utilise des règles explicites de réparation. Il est basé sur un raisonnement à base de cas.

Exemple : Réparation avec des règles spécifiques

Dans le cas du problème de réparation présenté page 40, on suppose que l'aléa d'un nouvel obstacle détectée a été identifié comme étant possible au moment de la description du problème.
Dans ce cas de figure, une règle particulière a été écrite. Par exemple une fois un obstacle détecté, on identifie dans le plan les mouvements y menant et en partant. Pour chaque couple d'actions passant par le nouvel obstacle, on le remplace par une série de mouvements qui relie les mêmes points de départ et d'arrivée mais sans passer par ce nouvel obstacle.
Cela permet de résoudre cet exemple simple. Mais pour être robuste aux différents scénarii possibles, il faut aussi traiter le cas où un objectif à explorer se situe sur un obstacle, où le mouvement vers l'obstacle est la dernière action du plan, où il n'existe pas de chemin de contournement, etc. Cela nécessite d'être exhaustif par rapport à tous les aléas auxquels on veut réagir.

2.2.4 Rejouer le raisonnement

Plutôt que d'utiliser uniquement un plan comme point de départ de la réparation, certains algorithmes utilisent aussi explicitement le raisonnement qui a mené jusqu'à ce plan. Cela nécessite de stocker plus d'information durant la recherche pour pouvoir les réutiliser pendant la réparation. On peut par exemple stocker pourquoi une action a été rajoutée dans le plan, ou quelles contraintes n'ont mené qu'à des plans invalides pour ne pas les réessayer durant la réparation.

RepairSHOP [Warfield et al. 2007] est une extension de l'algorithme SHOP, un algorithme de recherche HTN qui se base sur des méthodes totalement ordonnées (cf. section 1.3.1) pour rejouer le raisonnement.

C'est aussi le cas de PANDA [Bidot, Schattenberg et Biundo 2008], un algorithme de planification hybride (cf. section 1.3.2), qui pour réparer essaye de rejouer au plus proche les étapes de la recherche ayant mené au plan courant.

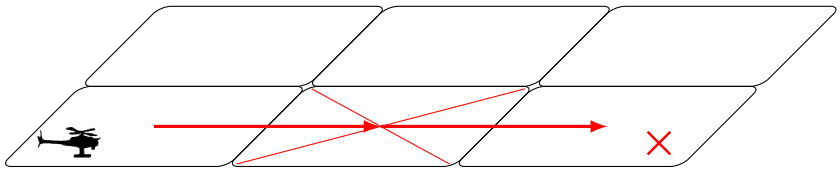
CAPlan/CBC [Muñoz-Avila et Weberskirch 1996] part d'une base de données de plans et commence par en déterminer le plus prometteur. Ensuite, il rejoue les décisions qui ont menées à l'élaboration du plan et n'explore pas les branches qui ont déjà été marquées comme des impasses durant la recherche initiale.

Exemple : Rejouer le raisonnement

Dans le cas du problème de réparation présenté page 40, on suppose que le plan initial et le raisonnement y ayant mené ont été conservés. Dans cet exemple, on suppose que les étapes ont été les suivantes :

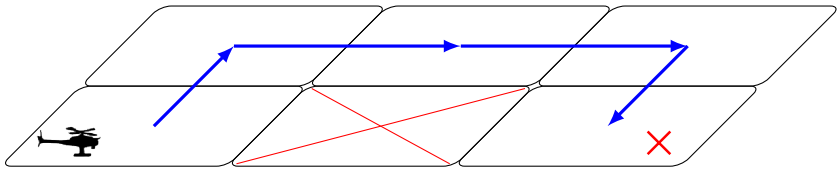
- Pour explorer le but, il faut ajouter une action d'observation
- Pour faire cette observation, il faut se rendre sur place
- Le chemin le plus court pour y aller est la ligne droite

Figure 2.4 – Exemple de réparation par réutilisation du raisonnement.



(a) Problème à réparer : le plan actuel n'est plus faisable.

Une fois le nouvel obstacle détecté, le raisonnement ayant mené au plan est rejoué. Dans ce cas, les deux premières étapes sont inchangées. La dernière étape par contre est différente : le nouvel obstacle change le chemin le plus court pour pour atteindre la zone d'observation. Il suffit alors de re-calculer ce chemin pour obtenir un plan faisable, ce qui mène à la solution suivante :



(b) Problème réparé.

2.3 Alternatives à la réparation

2.3.1 Planification conditionnelle

Au lieu de devoir réparer lors de l'arrivée d'un événement imprévu, une technique consiste à prévoir leur arrivée dès la planification. Dans ce cas, le plan produit ne contient pas seulement une séquence d'actions, mais plutôt un ensemble de branches. Chaque branche est associé à certaines conditions. C'est ce que l'on appelle la planification conditionnelle (*conditional planning*) [Peot et D. E. Smith 1992]. Cela revient à laisser des choix ouverts dans le plan et à calculer tous les plans possibles résultant de ces choix pour garantir que tous les plans seront valides.

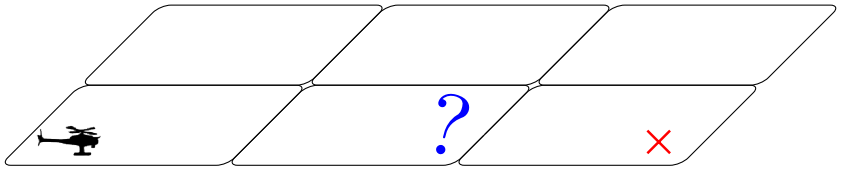
Par exemple dans le cas où il est possible (mais pas certain) qu'une route soit bloquée, le plan produit peut spécifier qu'après un déplacement et une observation le plan a deux branches. Soit la route menant au but est libre, et le robot l'emprunte. Soit elle est bloquée, et dans ce cas le robot contourne l'obstacle.

Cela permet de surmonter certains événements aléatoires sans avoir à réparer (toutes les recherches étant faites avant l'exécution). Néanmoins, il est souvent difficile de prévoir tous les événements possibles et le plan conditionnel prenant en compte tous ces événements possibles peut être trop gros pour être utilisable.

Exemple : Planification conditionnelle

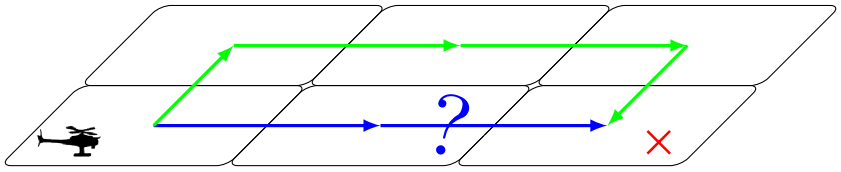
Dans le cas du problème de réparation présenté page 40, on suppose qu'avant l'exécution, la zone dont la traversabilité n'est pas connue a été identifiée. On cherche alors un plan capable de résoudre le problème dans toutes les situations et en prenant le chemin le plus court disponible.

Figure 2.5 – Exemple de planification conditionnelle.



(a) Problème à résoudre.

Le plan calculé est alors conditionnel : en fonction du résultat de la première action on effectue plutôt le plan en bleu (si aucun obstacle n'est détecté) ou le plan en vert. Cela permet d'éviter de réparer le plan actuel si un obstacle est détecté à l'endroit où il est attendu. Par contre, la réparation est inévitable si l'événement qui survient n'était pas attendu.



(b) Plan résultant de la planification conditionnelle : deux chemins sont possibles en fonction de l'évolution de l'exécution du plan.

2.3.2 Planification probabiliste

Au lieu de représenter les actions comme ayant des effets déterministes, la planification probabiliste propose de les représenter comme des effets probabilistes. Par exemple, une action de déplacement peut dériver et/ou échouer. La position du robot après son déplacement est donc représentée comme une distribution de probabilité. Cela permet d'éviter de réparer le plan courant si une action n'a pas produit ses effets nominaux.

Un modélisation possible se base sur les processus décisionnels Markovien (*MDP* en anglais) [Puterman 1994]. Le monde est représenté comme un ensemble d'états (cf. section 1.1.1) et les actions font passer d'un état à un autre. Mais ici les effets sont probabilistes : l'état après application d'une action est défini comme une distribution probabiliste sur l'ensemble des états atteignables. Si l'on suppose que l'état courant est complètement observable, on peut définir une *politique* plutôt qu'un plan : pour chaque état possible on définit quelle action appliquer plutôt qu'une succession d'actions. Certaines politiques peuvent même être aléatoires : à un état donné correspond une distribution d'actions possibles et l'action exécutée est choisie au hasard au moment de l'exécution. On cherche alors à maximiser une fonction cumulative qui dépend de l'état et de l'action appliquée à

chaque étape. On cherche à maximiser son espérance sur toutes les réalisations possibles de la politique. Une extension à PDDL (cf. section 1.1.3) nommé PPDDL [Younes et Littman 2004] a été proposée à l'occasion des compétitions internationales de planification. Plusieurs planificateurs permettent de faire de la planification probabiliste, tel que LAO* [Hansen et Zilberstein 2001] ou UCT [Browne et al. 2012].

Une extension de ce modèle propose de lever l'hypothèse d'observabilité de l'état courant. On suppose alors que les senseurs disponibles ne sont pas parfaits et qu'ils ne permettent pas d'avoir l'information exacte de l'état courant. On parle alors de processus décisionnel Markovien partiellement observables (*POMDP* en anglais). Dans ce cas, le système maintient une croyance sur l'état actuel : une distribution de probabilités sur l'état actuel. La politique est alors définie sur cette croyance. Ce modèle a été formalisé dans [Smallwood et Sondik 1973]. Des méthodes approchées permettant de résoudre des problèmes non triviaux ont été proposées [Pineau, Gordon et Thrun 2003 ; Silver et Veness 2010 ; T. Smith et Simmons 2004] et mises en œuvre durant les IPC dans la branche probabiliste.

D'autres modélisations se rapprochent plus de la recherche dans l'espace des plans (cf. section 1.2.3). Mais les fluents sont alors associés à une probabilité, et le but est d'atteindre un état but avec une certaine probabilité. C'est le cas de BURIDAN [Kushmerick, Hanks et Weld 1995] par exemple.

Ces représentations permettent de ne pas avoir à réparer le plan quand un effet non nominal se produit : il est intégré à la distribution de probabilités et le plan y est donc robuste. Mais le prix à payer est une complexité plus importante de la recherche, ce qui réduit la taille des problèmes solvables. De plus, il est nécessaire de connaître la distribution de probabilités des effets d'une action (ou d'en avoir une bonne approximation). Cette information est souvent indisponible et une approximation grossière est nécessaire, ce qui peut rendre les plans non adaptés au problème réel à résoudre.

Néanmoins, cette méthode permet de fournir des garanties en terme de probabilité d'atteinte d'un ensemble état ce qui la rend adaptée à des études comme celles de *model checking*. Dans le domaine aéronautique par exemple, on cherche à garantir une probabilité de panne critique inférieure à un certain seuil. Étant donné la probabilité de défaillance des systèmes de bord ainsi que des systèmes de secours, on peut vérifier l'absence d'une politique menant à une faille avec une probabilité donnée.

Elle est aussi adaptée dans le cas où les actions sont naturellement non déterministe et où une planification déterministes induirait des réparations trop fréquentes.

2.4 Interfaçage planification/exécution

En plus de la résolution des problèmes de réparation, l'interaction entre l'exécution des plans et leur réparation est un domaine actif de recherche. L'approche classique, qui consiste à réparer le plan en cours d'exécution, peut conduire à un blocage temporaire du système pendant le temps de la réflexion. Elle nécessite de plus que le planificateur utilisé puisse utiliser les informations issues de l'exécution : les dates précises des actions passées, la distinction entre les actions qui ont été commencées et celles qui ne le sont pas, etc.

Ainsi, des travaux ont porté sur la réparation d'un plan pendant l'exécution de celui-ci. IxTeT-eXeC [Lemai-Chenevier 2004] propose de faire intervenir la planification et l'exécution sur les mêmes données. Les plans sont représentés par des timelines (cf. section 1.1.6) avec une gestion des ressources. En cas de problème, l'exécution de la partie valide du plan continue tant que possible pendant que le plan est réparé. De plus, le planificateur

est autorisé à abandonner certains buts pour continuer la poursuite des autres.

Un problème proche est celui de la planification continue : on s'intéresse à un système dont l'horizon de planification ne suffit pas à accomplir une mission, qui peut être éventuellement infinie. C'est le cas par exemple pour certains satellites : il faut prévoir en continu les nouvelles acquisitions au fur et à mesure de la vie du satellite et il est impossible de faire un plan global de toutes les acquisitions au lancement du satellite. L'agent doit alors mettre à jour son plan périodiquement en augmentant son horizon de planification et en incorporant éventuellement de nouveaux buts. C'est le principe de l'horizon glissant. CASPER [Chien et al. 2000] propose une méthode pour réaliser une réparation itérative et synchronisée avec l'exécution du plan pendant son exécution, en augmentant à chaque fois un peu l'horizon et en gardant si possible les tâches qui ont déjà été prévues. De plus, une hiérarchie de plans sont considérés, avec un niveau de détail et un horizon différent. T-REX [McGann et al. 2008] propose une autre architecture basée sur une hiérarchie de planificateurs et d'exécuteurs ayant un horizon de planification et un niveau de détail différents : les actions d'un niveau correspondent aux buts du niveau inférieur. Le planificateur utilisé est EUROPA [Jónsson et al. 2000] et le plan est basé sur une représentation en timeline.

Conclusion

Dans un système robotique réel, le plan calculé en début de mission est rarement celui qui est vraiment réalisé. Le plan doit être mis à jour pendant l'exécution pour prendre en compte les événements aléatoires, attendus ou non, pouvant survenir. Cette mise à jour est appelée *réparation* quand son but est de modifier le plan courant pour l'adapter à la nouvelle situation (par opposition à la *replanification* qui ne tient pas compte du plan courant).

Ce problème de *réparation* est plus formellement décrit à la section 2.1. On y présente aussi son intérêt et les métriques particulières permettant de l'évaluer.

La section 2.2 présente différentes familles d'algorithmes de réparation. Ces algorithmes peuvent être plus ou moins proches des algorithmes de planification vus au chapitre 1 et nécessitent souvent des connaissances ou des heuristiques supplémentaires.

La section 2.3 présente des alternatives à la réparation. Les plans produits sont alors robustes à certains événements donnés, ce qui limite le nombre de réparations nécessaires, mais nécessitent une connaissance plus fine des différents événements aléatoires qui peuvent survenir.

Finalement, la section 2.4 présente certaines solutions qui ont été proposées pour interfacer au mieux la réparation avec l'exécution. Cela permet de diminuer l'impact des temps de réparation ou de réaliser une mission en continu sur un temps bien supérieur aux horizons de planification.

Deuxième partie

Développement d'un algorithme de planification, d'exécution et de réparation

3

Choix d'un modèle de planification et développement de l'algorithme HiPOP

Sommaire

Pour parvenir à notre objectif d'autonomie collective d'une équipe de robots autonomes individuellement, la première étape est d'avoir un algorithme de planification ayant les caractéristiques suivantes :

- Résolution d'une variété de problèmes pour être le plus versatile possible et donc s'adapter à de potentielles évolutions des problèmes à résoudre ;
- Durée de calcul raisonnable pour des problèmes types de surveillance d'une zone par une équipe de robots hétérogènes ;
- Gestion des échéances pour permettre à chaque robot de respecter ses rendez-vous avec d'autres robots ;
- Production d'un plan facilitant son exécution en ligne ;
- Extensibilité facilitée pour permettre la réparation d'un plan.

Nous allons donc nous intéresser dans ce chapitre au développement d'un tel algorithme de planification et en expliquant pourquoi les algorithmes existants ne répondent pas à nos besoins. Nous allons d'abord choisir un cadre de modélisation qui sera utilisé pour la description par l'opérateur humain des problèmes à résoudre et pour le raisonnement du planificateur (section 3.1). Ce cadre servira de base au développement l'algorithme de planification (section 3.2). Finalement, l'algorithme sera validé en vérifiant ses performances sur des problèmes variés issus des compétitions de planification IPC et sur des problèmes de recherche multiagents (section 3.3).

3.1 Choix d'un modèle de planification

Avant de développer un nouvel algorithme, nous devons choisir parmi les différents cadres de modélisation et les différentes familles d'algorithmes présentés au chapitre 1 lesquels utiliser.

Choix de la famille d'algorithme

L'absence potentielle de communication entre les robots, en particulier lorsqu'un problème survient et qu'une réparation doit avoir lieu, impose que chaque robot soit en mesure de réparer sa partie du plan. De plus, si un problème survient et isole un robot du reste de l'équipe (par exemple un obstacle bloque à la fois le robot et ses communications, l'antenne a subi une casse ou une panne, etc.), il est intéressant que le reste de l'équipe puisse le détecter. Pour cela, nous avons choisi d'introduire des rendez-vous entre les robots pendant lesquels la communication est supposée réalisable (les robots étant proches avec une ligne de vue directe). Ces rendez-vous deux à deux et définis géographiquement et temporellement (à une date donnée) vont permettre à un robot de réagir lorsque l'autre robot est en retard. Cela impose donc que chaque robot soit capable de respecter des échéances et donc de calculer des plans respectant ces échéances. Notre algorithme doit donc être capable de raisonnement temporel.

Au vu de ces contraintes, la recherche dans l'espace des plans semble la plus adaptée (cf. section 1.2.3). Non seulement cette technique est capable de raisonnement temporel, mais en plus les plans produits sont flexibles temporellement. Cela signifie qu'une action n'est pas uniquement caractérisée par une date précise à laquelle elle doit avoir lieu mais par un ensemble de contraintes temporelles qui permettent de choisir en cours d'exécution la date du début de l'action. Les plans sont ainsi robustes à un certain nombre de retards impondérables lors de l'exécution d'une action, et la structure du plan permet de détecter si une action retardée aura ou non une influence sur l'accomplissement de la mission. Il a aussi été montré que cette flexibilité permet de fusionner plus facilement des plans concernant différents robots [Hashmi et Seghrouchni 2010]. Cette caractéristique semble adaptée à nos problèmes multiagents, où le plan contient explicitement les contraintes/interactions entre les actions des différents robots.

Néanmoins, cette technique n'est pas très rapide et pourrait mettre trop de temps à résoudre un problème de la taille de ceux que nous souhaitons résoudre (une dizaine de robots et quelques dizaines de fluents dans le but). De plus, l'objectif de ce travail est de produire un algorithme qui sera utilisé dans des scénarios réels avec de vrais engins, ce qui impose des contraintes difficilement modélisables et rendant le problème potentiellement beaucoup plus complexe (des contraintes sur les trajectoires réalisables par les robots par exemple). Pour pallier ces problèmes, une technique utilisant des actions hiérarchiques lors d'une recherche dans l'espace des plans a été proposée sous le nom de planification hybride (cf. section 1.3.2). Elle permet d'accélérer la recherche et d'imposer des contraintes sur la forme des plans solutions produits. Nous avons donc décidé d'utiliser la planification hybride pour notre algorithme.

Cette technique ne nécessitant pas une hiérarchie complète pour résoudre un problème (elle permet par exemple l'ajout opportuniste d'actions pour satisfaire un but et d'adapter les méthodes au plan en cours de résolution), il est possible de contrôler la quantité d'information supplémentaire fournie au planificateur. Cela nous permettra aussi d'étudier l'impact de ces actions hiérarchiques sur le résultat pour valider la pertinence de ce choix. Cela permet aussi de limiter l'expertise humaine nécessaire et d'être plus tolérant à un

changement dans la modélisation des actions hiérarchiques, un des problèmes majeurs d'une planification purement hiérarchique.

Choix de la représentation du problème

Notre volonté de pouvoir résoudre plusieurs types de problèmes nous a naturellement orienté vers l'utilisation du langage de description PDDL (cf. section 1.1.3). L'avantage principal de ce langage est l'existence d'un ensemble important de problèmes, de planificateurs et d'un vérificateur de plans [Howey et Long 2003]. Les compétitions des IPC, organisées en parallèle de la conférence ICAPS, ont rendu disponible beaucoup de planificateurs ayant concouru dans les différentes catégories et les domaines utilisés.

Le choix du couple planification hybride/PDDL a un inconvénient majeur : le langage PDDL n'est pas prévu initialement pour pouvoir représenter des actions hiérarchiques. Cela signifie qu'il est nécessaire de proposer une extension pour pouvoir représenter ces actions, mais aussi qu'aucune action hiérarchique n'est définie dans les domaines des IPC.

Ce problème n'est pas nouveau et tous les planificateurs hybrides ont dû trouver une manière de représenter les problèmes permettant de décrire ces actions hiérarchiques. PANDA [Schattenberg 2009] a fait le choix d'utiliser un langage nouveau, au format XML. L'inconvénient est qu'il n'existe aucun problème ou planificateur de référence et qu'il a fallu créer un tout nouveau langage. FAPE [Dvorak et al. 2014] a fait le choix de l'utilisation d'ANML (cf. section 1.1.7), un langage générique mais encore trop nouveau pour posséder un ensemble de problèmes, de planificateurs ou de vérificateur de plans. Nous avons choisi d'étendre PDDL pour pouvoir bénéficier de tout l'écosystème déjà existant et de garder une compatibilité maximale avec une syntaxe de PDDL classique. Cela nous permet d'être capable de raisonner sur tout problème PDDL comme un algorithme de recherche dans l'espace des plans et d'être capable d'utiliser toute action hiérarchique supplémentaire pour faire une recherche hybride. L'inconvénient de cette approche est la faiblesse relative de la puissance de description temporelle de PDDL, surtout si nous voulons garder une rétro-compatibilité.

Choix d'une recherche relâchée ou non

Un dernier choix doit être fait : raisonner avec les fluents complètement instanciés ou non ? Le raisonnement avec tous les fluents instanciés consiste à transformer chaque action décrite dans le domaine en une multitude d'actions possibles. Le planificateur considère alors toutes ses actions indifféremment : les actions `move a c` (un déplacement de `a` vers `c`) et `move b c` sont complètement séparées. La recherche relâchée (qui consiste à garder les variables dans la description des actions, cf. page 29) permet de raisonner avec des actions comme `move ?x c` qui représente n'importe quelle action de déplacement se finissant en `c`. Cela implique de garder un ensemble de contraintes d'instanciation pour chaque plan et l'ajout de nouvelles contraintes peut servir à résoudre un défaut ou à éliminer un plan si ses contraintes d'instanciation ne sont pas cohérentes.

L'avantage de produire des plans où certaines actions ne sont que partiellement instanciées ne nous a pas paru aussi important que celui d'avoir des contraintes temporelles souples. En effet lors de l'exécution il est difficile de pouvoir faire des choix plus éclairés que lors de la planification sur quelles actions réaliser en limitant la puissance de calcul nécessaire (pour ne pas refaire une planification en cours d'exécution). Les plans produits pouvant difficilement être flexibles sur les positions à atteindre, seul le choix des robots à

utiliser pour telle ou telle action semble intéressant. Mais dans un contexte où les communications peuvent être interrompues, laisser ce choix être fait pendant l'exécution nous a semblé inadéquat. A l'inverse, les durées des actions sont souvent hors de contrôle et un plan capable de s'y adapter est utile pour limiter le besoin de réparation, ce choix ne pouvant pas être fait à priori.

Il restait alors des considérations de performance et de simplicité d'utilisation pour décider. La recherche complètement instanciée est plus simple d'utilisation (bien que plus gourmande en mémoire) : il n'est pas nécessaire de maintenir un ensemble de conditions d'instanciation et de vérifier sa cohérence ou de manipuler des actions partiellement instanciées. De plus, l'évaluation par une heuristique du coût d'un lien ouvert non instancié est beaucoup moins évidente que celle d'un lien ouvert instancié : si l'on ne sait pas encore de quoi on a besoin il est difficile d'estimer avec précision le nombre d'actions nécessaires. La grande majorité des planificateurs présents dans la littérature utilisent donc donc une représentation totalement instanciée [Helmert 2009], et c'est le choix que nous avons fait.

3.2 Conception d'un algorithme de type planification hybride

Comme il n'existe pas à notre connaissance de planificateur hybride disponible satisfaisant les caractéristiques souhaitées, nous avons décidé en concevoir un. Pour cela, il a été nécessaire de définir le formalisme utilisé pour décrire les problèmes ainsi que les heuristiques qui guident la recherche. Avant de détailler ces choix, nous allons poser les définitions qui seront utilisées par la suite.

3.2.1 Définitions et rappel sur la planification hybride

On rappelle ici le principe de la planification hybride et on introduit des notations qui seront utilisées par la suite. La plupart des notations et des définitions sont tirées de la recherche dans l'espace des plans (cf. pages 32 et suivantes).

Les actions sont définies comme un mélange d'actions élémentaires et d'actions hiérarchiques (cf. définitions 15-16 page 21).

Définition 32 : Action (Hybride)

Soit F un ensemble de fluents, une action est définie par :

- Un nom unique dans le problème considéré ;
- Un ensemble $Pre \subseteq F$ de préconditions ;
- Un ensemble $Add \subseteq F$ d'effets ;
- Un ensemble $Del \subseteq F$ d'effacements ;
- Un ensemble $\mathcal{M}$ de méthodes. Chaque méthode m est représentée comme un plan partiel $\langle \mathcal{T}(m),\ CL(m),\ TL(m),\ H(m),\ Init(m),\ Goal(m) \rangle$ (cf. définition 33) avec $Pre \subseteq Init(m)$ et $Add \cup \neg Del \subseteq Goal(m)$.

Une action telle que $\mathcal{M} \neq \emptyset$ est appelée action abstraite (ou hiérarchique). Si $\mathcal{M} = \emptyset$, elle est élémentaire.

Une méthode étant censée représenter une manière de réaliser une action abstraite, elle doit pouvoir partir de ses préconditions et chercher à établir ses effets. Néanmoins, une méthode n'est pas forcément un plan solution : elle peut ne pas établir une partie des effets de l'action. De même, elle peut ne pas utiliser tous les éléments de $Init(m)$. Si $Del \neq \emptyset$, alors le but de chaque méthode doit aussi contenir les négations des fluents de Del (représenté ici par $\neg Del$).

Un plan reprend la même composition qu'en POP avec en plus la hiérarchie parmi les tâches introduites par les actions abstraites. La définition des liens causaux ne change pas (cf. définition 26).

Définition 33 : Plan partiel (Hybride)

Soit $\langle F, A, I, G\rangle$ un problème de planification hybride.
Un plan $\Pi = \langle \mathcal{T}, CL, TL, H, Init, Goal\rangle$ est défini par :

- Un ensemble de tâches $\mathcal{T} = \{(a_i, t_i^s, t_i^e)\}$ où a_i est une action de A et où les t_i^s, t_i^e représentent des instants temporels.
- Un ensemble de liens causaux CL, chaque lien $\langle \tau_s, t_s, \tau_e, t_e, f\rangle$ indique que la tâche τ_s crée le fluent f à l'instant t_s qui est garanti de rester vrai jusqu'à ce que la tâche τ_e finisse d'en avoir besoin en t_e.
- Un ensemble de liens temporels TL, chaque lien $\langle t_s, t_e\rangle$ indiquant une précédence $t_s \prec t_e$.
- Un ensemble de relations $H = \{(\tau_i, m_i, \tau_i^0, ..., \tau_i^n)\}$ où τ_i est une tâche associée à une action abstraite a_i, m_i est une méthode de a_i et les τ_i^j sont des tâches du plan. Chaque relation indique que les tâches $\tau_i^0 ... \tau_i^n$ ont été introduites par l'instanciation de la tâche τ_i par sa méthode m_i.

De plus, chaque plan contient deux tâches particulières : une correspondant aux conditions initiales (précédent toutes les tâches et avec $Add = Init$) et une correspondant aux buts (postérieure à toutes les tâches et avec $Pre = Goal$).
Un plan correspond au problème (et ne peut en être une solution) que si $Init = I$ et $Goal = G$.

Un défaut peut être de trois types : les liens ouverts et les menaces (voir définition 28 page 34) viennent de POP et les défauts abstraits sont introduits par la planification hybride.

Définition 34 : Défaut (Hybride)

Étant donné un plan hybride Π, on peut calculer l'ensemble des défauts $Flaws(\Pi)$.
Chaque défaut est d'un des trois types suivant :

- Lien ouvert : il n'existe pas de lien causal pour une précondition d'une tâche du plan.
- Menace : une tâche effaçant le fluent d'un lien causal peut arriver pendant ce lien causal.

– Abstrait : une tâche abstraite du plan n'est pas instanciée.

La résolution des deux défauts venant de POP ne change pas. Pour un lien ouvert, il faut ajouter un lien causal depuis une action existante ou depuis une nouvelle action (ce qui génère éventuellement de nouveaux liens ouverts). Pour une menace, il faut ajouter une contrainte d'ordonnancement entre l'action menaçante et le lien causal menacé : avant ou après. Pour un défaut abstrait, il faut instancier l'action abstraite en choisissant une méthode et en introduisant tous ses éléments dans le plan. On appelle *résolveurs* d'un défaut les plans permettant de résoudre un défaut donné.

L'algorithme de la recherche hybride (Algorithme 3) peut être directement obtenu de l'algorithme de recherche générique présenté au chapitre 1 page 28.

Algorithme 3 : Algorithme de planification hybride

Entrée : I, le nœud de recherche initial
$E = \{I\}$ `// Frontière d'exploration`
tant que $E \neq \emptyset$ **faire**
  Choix d'un plan $u \in E$ `// Choix d'un plan selon une heuristique`
  $E = E - \{u\}$
  Soit Δ l'ensemble des défauts de u
  si $\Delta = \emptyset$ **alors**
    retourner u
  Choix de $\delta \in \Delta$ `// Choix d'un défaut selon une heuristique`
  Soit B l'ensemble des plans partiels résolvant δ dans u
  $E = E \cup$ Élague(B) `// Retire les plans incohérents temporellement`
retourner Échec

Un algorithme hybride, comme tout algorithme de recherche dans l'espace des plans, permet de gérer des buts avec échéances. En effet, pour chaque but il suffit d'ajouter une action dans le plan avec pour seule précondition ce but. Des contraintes temporelles sont ajoutées dans le STN pour contraindre la date de début de cette action à la date fixée par l'échéance. On peut même garantir de maintenir un fluent vrai pendant une certaine durée en contraignant la date de la fin de l'action. Ces nouvelles actions et contraintes associées doivent être introduites dans le plan partiel initial I de l'algorithme 3.

3.2.2 Modélisation des actions hiérarchiques

Pour résoudre des problèmes de planification, il est tout d'abord nécessaire de pouvoir les décrire. L'utilisation du format PDDL permet d'avoir déjà un formalisme permettant de décrire les problèmes composés uniquement d'actions élémentaires. Pour pouvoir décrire des actions abstraites, nous proposons d'étendre PDDL.

Dans un algorithme de planification hybride, les actions abstraites sont considérées comme les actions élémentaires durant la recherche tant qu'elles ne sont pas instanciées : c'est pour cette raison que nous avons choisi d'étendre la définition des actions PDDL pour intégrer la description de méthodes. Un exemple de ce formalisme est fourni à la page 63, après l'explication des choix faits ci-dessous.

Chaque méthode est définie comme un plan partiel (cf. définition 27), *i.e.* une liste de tâches, une liste de liens causaux et une liste de contraintes temporelles entre ces tâches.

De plus, nous avons décidé de rajouter un ensemble de préconditions spécifiques à chaque méthode, à l'instar de ce qui peut se faire en planification HTN. Ainsi les fluents de $Init(m)$ sont les fluents de Pre de son action plus les fluents spécifiques à cette méthode. Par contre, nous n'avons pas permis d'ajouter des éléments dans les buts des méthodes : cela aurait été assez peu utile car avant l'instanciation ils n'auraient pas pu être utilisés. Donc $Goal(m) = Add \cup \neg Del$.

En effet, lors de l'introduction des actions hiérarchiques dans le plan, le planificateur n'utilise pas les différentes méthodes. Il se fie à la description de l'action pour savoir ce qu'elle permet de faire (via les effets et les effacements) et ce dont elle a besoin (via les préconditions). Après que l'action abstraite ait été introduite dans le plan se pose la question de la méthode choisie pour la réaliser. Autant il nous semble pertinent de devoir assurer de nouvelles conditions lors de l'instanciation, autant la création de nouveaux buts semble inutile. Si ces fluents doivent être utilisés par la suite (donc après l'instanciation de l'action abstraite), ils sont présents dans les effets des tâches de la méthode et sont donc disponibles pour le reste du plan. S'ils ne sont pas utilisés par la suite, il n'est pas nécessaire de forcer leur établissement.

Chaque tâche appartenant à une méthode est identifiée par un nom, unique pour cette méthode, et par l'action à laquelle elle correspond. Chaque lien causal est identifié par ses deux tâches et son fluent (les instants temporels concernés pouvant être retrouvés). Les contraintes temporelles étant utilisées pour éviter de potentielles menaces, il est suffisant de pouvoir introduire des précédences entre les tâches et pas une durée fixe. Les contraintes temporelles sont donc uniquement définies par deux tâches et imposent que la première arrive avant la seconde. Il ne serait néanmoins pas plus difficile d'autoriser l'utilisateur à spécifier une durée minimum et/ou maximum entre ces actions : le STN le permettrait sans modifications particulières.

Bien que cette description soit suffisante pour définir les méthodes, elle n'est pas suffisante pour garantir que l'utilisation de ces méthodes n'introduira pas de nouveaux défauts tels que ceux décrits ci-dessous. La planification hybride peut fonctionner dans de tels cas mais cela induit un effort de recherche au planificateur pour résoudre des défauts que la description hiérarchique pourrait empêcher. Nous avons donc enrichi la description des actions hiérarchiques pour permettre l'écriture de méthodes qui n'introduisent pas de nouveau défaut lors de leur utilisation.

Le premier type de défaut considéré est le lien ouvert : si toutes les préconditions d'une tâche introduite ne sont pas associées avec des liens causaux, alors un lien ouvert doit être introduit. La description décrite jusqu'à présent permet de décrire les liens causaux entre tâches filles, mais ne permet pas d'établir des liens causaux avec des tâches extérieures à la méthode. Nous avons donc nommé les deux tâches particulières appartenant à tous les plans partiels (`:init` et `:goal`) pour permettre d'établir des liens causaux avec elles. Ainsi les méthodes peuvent utiliser les fluents qui sont garantis pour l'action abstraite (*i.e.* ceux appartenant à Pre) et peuvent garantir les effets de l'action abstraite, ce qui fait le lien avec les tâches extérieures. Avant l'instanciation, des liens causaux seront donc présents entre les tâches extérieures et l'action abstraite. Après l'instanciation, des nouveaux liens causaux pourront faire le lien entre les préconditions/effets de l'action abstraite et les tâches de la méthode.

Le deuxième type de défaut est la menace : il faut pouvoir empêcher l'action abstraite de survenir en même temps que certains liens causaux. Pour les actions élémentaires, cette

détection se fait avec les effacements de l'action : une action ne peut pas survenir durant un lien causal qui correspondrait à un de ses effacements. Malheureusement, le cas des actions abstraites n'est pas si simple. Les descriptions de ses préconditions et de ses effets ne sont pas suffisantes pour savoir quels seront les fluents qui seront supprimés durant l'exécution d'une de ses méthodes. Par exemple une action qui consisterait à déplacer un robot et à le replacer à son point de départ ne pourrait pas se distinguer d'une action qui laisserait le robot immobile. Même si une description exhaustive des fluents menacés n'est pas nécessaire, elle permet au planificateur de se rendre compte bien plus tôt dans la recherche (*i.e.* avant d'instancier les actions abstraites) qu'un plan n'est pas faisable. Nous avons donc ajouté un élément à chaque action abstraite : la liste des *conflits*.

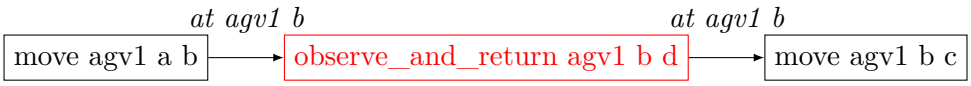
Cette liste représente la liste des fluents avec lesquels l'action abstraite non instanciée sera considérée comme étant en menace. Pour simplifier la description du problème, nous avons permis l'utilisation du symbole `*` pour décrire toute une famille de fluents avec un unique fluent. Ainsi le conflit `at ?r *` indique qu'une action est incompatible avec tout lien causal sur la position du robot `r` sans avoir besoin d'énumérer toutes les positions possibles. Cela indique qu'on ne peut pas utiliser une action de déplacement du robot `r` antérieure à l'action hiérarchique pour assurer une position du robot nécessaire après l'action hiérarchique.

De plus ces conflits sont aussi utilisés pour détecter l'incompatibilité entre deux actions hiérarchiques : si elles partagent un conflit alors elles doivent être séparées (l'une doit arriver avant l'autre). Cela permet d'éviter les menaces apparaissant suite à l'instanciation de ces deux actions.

Exemple : Description d'une action abstraite

En reprenant le type de problème présenté au chapitre 1, on peut décrire une action permettant d'explorer une cellule autre que la cellule dans laquelle se situe le robot en lui faisant faire un aller-retour si nécessaire. Un exemple de plan contenant cette action non instanciée est le suivant :

Figure 3.1 – Exemples d'instanciation d'une action abstraite.



(a) Partie d'un plan contenant une action abstraite non instanciée.

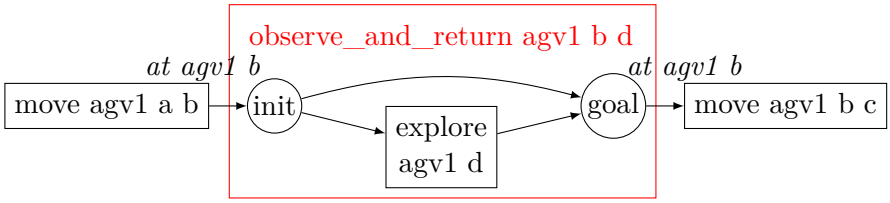
L'action peut alors être décrite comme suit :

```
(:action observe_and_return
  :parameters (?r - robot ?from ?to - loc)
  :conflict-with (at ?r *)
  :precondition (at ?r ?from)
  :effect (and (at ?r ?from) (explored ?to) )
  :methods (
    :method same_loc
    :actions (exp (explore ?r ?to))
    :precondition (= ?from ?to)
    :causal-links (:init exp (at ?r ?from))
```

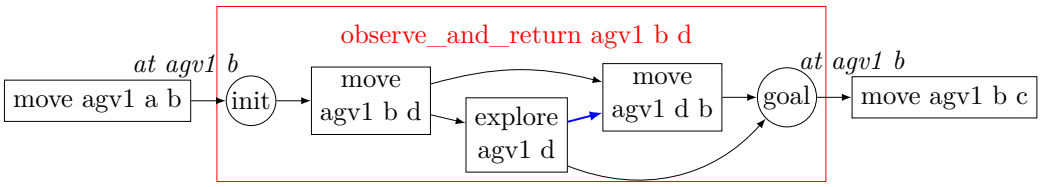
```
                    (exp :goal (explored ?to))
                    (:init :goal (at ?r ?from))
    :temporal-links

    :method different_loc
    :actions (move1 (move ?from ?to))
             (exp (explore ?t ?to))
             (move2 (move ?to ?from))
    :precondition (not (= ?from ?to))
    :causal-links (:init move1 (at ?r ?from))
                  (move1 exp (at ?r ?to))
                  (move1 move2 (at ?r ?to))
                  (exp :goal (explored ?to))
                  (move2 :goal (at ?t ?from))
    :temporal-links (exp move2)
  )
)
```

Cette action propose deux méthodes : une dans le cas où le robot est déjà sur place et une dans le cas où il doit faire l'aller-retour. La différence est faite grâce aux préconditions de chaque méthode. Dans les deux cas, les deux effets de l'action sont garantis par un lien causal. Dans la deuxième méthode, il est nécessaire d'indiquer qu'il faut attendre d'avoir fini l'exploration de la cellule avant de retourner à son point de départ : c'est le rôle du lien temporel.

L'instanciation se passe alors comme suit :



(b) Partie d'un plan contenant une action abstraite instanciée dans le cas où $d = b$. Les liens causaux sont indiqués en noir.



(c) Partie d'un plan contenant une action abstraite instanciée dans le cas où $d \neq b$. Les liens causaux sont indiqués en noir. La contrainte temporelle de précédence est indiquée en bleu.

Les liens causaux de l'action abstraite non instanciée correspondent directement à des liens causaux pour les actions de début et de fin de la méthode (qui ont les mêmes préconditions et effets).

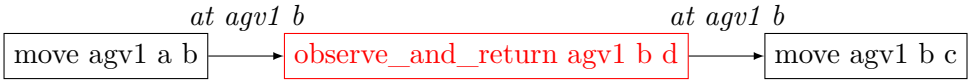
Cette méthode de description permet donc de représenter des méthodes n'introduisant pas (ou peu) de nouveaux défauts. Si un plan n'a que des défaut abstraits, l'instanciation d'une action abstraite par un méthode n'introduisant pas de nouveaux défauts produira toujours un plan dont les seuls défauts sont abstraits.

Une méthode vérifiant cette propriété doit donc garantir qu'aucun lien ouvert ne sera introduit, ni aucune menace entre deux éléments de la méthode. De plus, tout conflit entre un lien causal extérieur à la méthode et une action de la méthode doit aussi être détecté en amont comme une menace entre l'action abstraite non instanciée et le lien causal. Les seuls cas de figure pouvant alors résulter en l'ajout d'une menace sont ceux d'une menace d'une action extérieure à la méthode sur un lien causal interne à la méthode (voir la figure 3.2d). En pratique ce cas de figure ne s'est jamais présenté : une action qui efface un fluent en a généralement besoin en précondition. Il est alors possible de détecter une menace entre l'action hiérarchique et le lien causal d'entrée grâce aux conflits.

Exemple : Menaces impliquant des actions abstraites

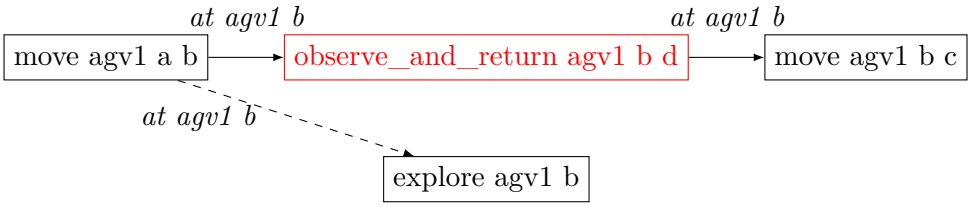
En reprenant l'exemple présenté page 63, imaginons un plan partiel comme suit :

Figure 3.2 – Exemples de détection de menaces par des actions abstraites.



(a) Partie d'un plan partiel contenant une action abstraite non instanciée.

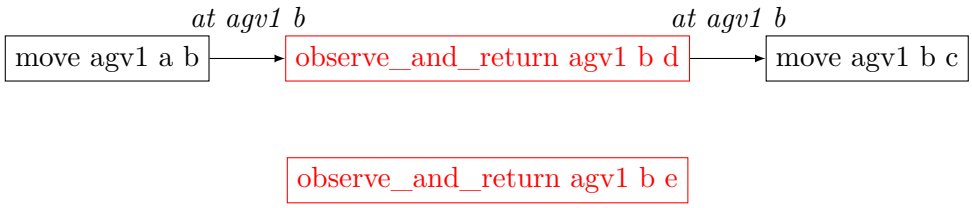
L'action abstraite *observe_and_return agv*1 *b d* est en conflit avec tous les fluents de la forme *at agv*1 ∗. Si on y introduit une autre action de mouvement pour l'*agv*1, aucune menace n'est détectée. Par contre, un conflit apparaîtra dès qu'un lien causal sera établi. Cette situation est représentée sur la figure 3.2b : l'introduction du lien causal pointillé permet de détecter la menace



(b) Partie d'un plan contenant une action abstraite non instanciée et une autre action de déplacement concurrente sans lien ouvert. Une menace est détectée quand le lien causal en pointillé est introduit.

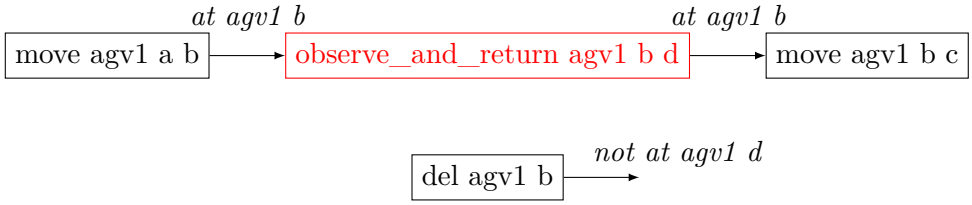
Pour accélérer la détection de ces menaces, il est aussi possible d'introduire des conflits entre une action hiérarchique et une action élémentaire en utilisant ses préconditions. Si une précondition d'une action est un des conflits d'une action abstraite, on considère une menace entre cette action abstraite et un futur lien causal : la résolution de ce défaut implique de séparer temporellement ces deux actions avant l'instanciation de l'action abstraite.

Si une autre action hiérarchique est introduite (cf. figure 3.2c), les conflits des deux actions vont être utilisés pour permettre de détecter une incompatibilité entre ces actions. La menace ainsi détectée implique aussi de séparer temporellement ces deux actions.



(c) Partie d'un plan contenant une action abstraite non instanciée et une autre action abstraite partageant au moins un conflit. Une menace particulière est détectée entre deux actions abstraites.

Par contre, ce mécanisme ne permet pas de détecter des menaces entre l'action hiérarchique et une action qui effacerait un fluent utilisé par un lien causal d'une méthode de l'action abstraite (cf. figure 3.2d). Il faudrait attendre l'instanciation de l'action pour détecter le conflit. Néanmoins, si cette action qui efface un fluent en a besoin en précondition, alors la menace serait détectée.



(d) Partie d'un plan contenant une action abstraite non instanciée et une autre action élémentaire. Cette action est telle que $Pre = Add = \emptyset$ et $Del = \{at\ agv1\ d\}$.

On cherche alors à caractériser ces méthodes qui n'introduisent pas de nouveaux défauts et ces plans n'ayant que des défauts abstraits.

Définition 35 : Plan abstrait

Un plan abstrait est un plan (Π) tel que tous ses défauts sont abstraits : $\forall \delta \in Flaws(\Pi), \delta$ est abstrait.

Un plan abstrait est un plan dans lequel toutes les actions ont leurs préconditions respectées (y compris le but) et aucun lien causal n'est menacé. On parle aussi de plan à haut niveau. En effet, intuitivement cela revient à dire que l'on a la forme générale d'un plan valide, mais qu'il reste encore certains détails et certaines transitions à établir.

Pour ces méthodes permettant de limiter la recherche tel que décrit ci-dessus, on les appelle *méthodes complètes*. Pour plus de généralité, on autorise une méthode complète à avoir des défauts abstraits. Si toutes les actions abstraites possèdent des méthodes complètes (et qu'il n'y a pas de cycles possibles lors de l'instanciation), la propriété globale reste la même : étant donné un plan abstrait on est garanti de pouvoir trouver une solution sans avoir à explorer beaucoup de nouveaux plans, il faudra éventuellement enchaîner des

instanciations. Mais à chaque instanciation, on est sûr que le plan fournira une solution au problème. Pour qu'une méthode soit complète, il faut donc qu'elle n'introduise pas de lien ouvert ou de menace lors de son instanciation (*i.e.* qu'elle corresponde à un plan abstrait) et qu'elle ait déjà séparé temporellement l'action abstraite de toutes les autres actions pouvant menacer un de ses éléments. Il faut donc s'assurer que les conflits de la méthode permettent de détecter toutes ces menaces avant l'instanciation. On aboutit alors à la définition suivante :

Définition 36 : Méthode complète

Soit m une méthode de l'action a.
m est complète ssi. m est un plan abstrait et $\forall \tau \in \mathcal{T}(m), \forall f \in Del(\tau), f$ est dans les conflits de a.

Définition 37 : Action complète

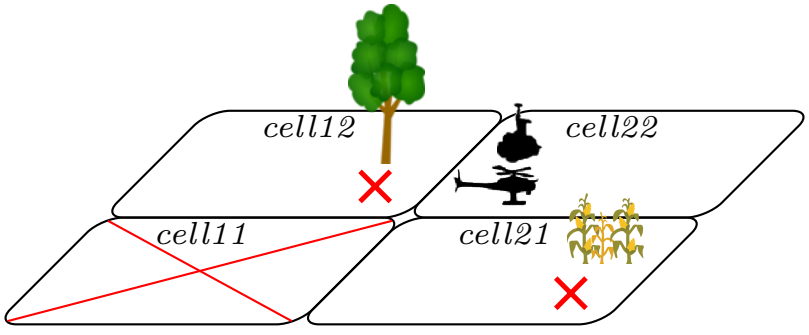
Une action est complète si toutes ces méthodes sont complètes.

Avoir un domaine composé uniquement d'actions élémentaires ou d'actions complètes n'est pas nécessaire à la bonne résolution d'un problème. Néanmoins, cela facilite la recherche en limitant le temps de recherche nécessaire une fois qu'un plan abstrait est trouvé.

Exemple : Recherche hybride

En reprenant le problème de l'exemple de la page 8, rappelé figure 3.3a, on montre sa résolution à l'aide d'un algorithme de recherche hybride. On suppose que les actions abstraites disponibles sont les *observe_and_return* présentés dans l'exemple précédent (page 63).

Figure 3.3 – Exemple de résolution d'un problème à l'aide d'une recherche hybride.

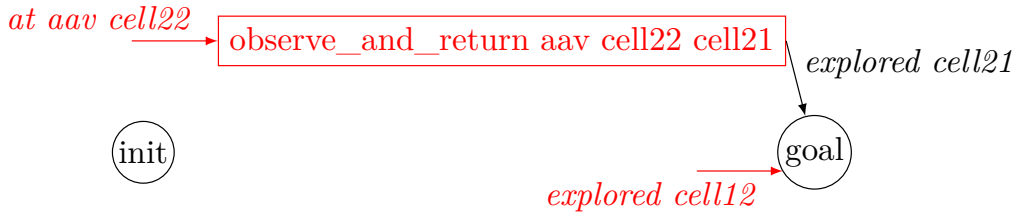


(a) Représentation du problème à résoudre.

Le recherche commence par un plan vide, constitué uniquement des actions initiale et finale :

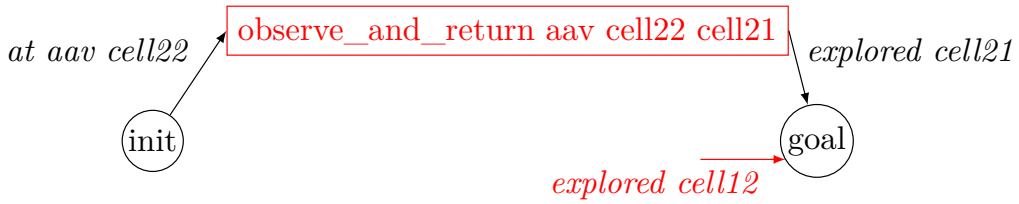
(b) Plan vide.

Ce plan contient deux liens ouverts, représentés en rouge. Pour résoudre le premier lien ouvert, le planificateur peut ajouter une action abstraite. On obtient alors le plan suivant :



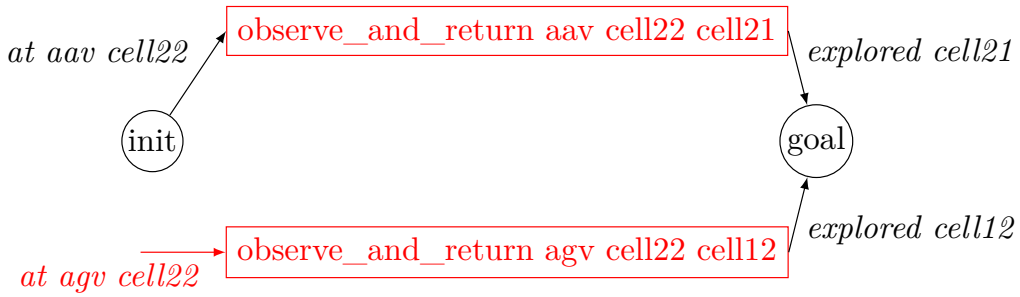
(c) Plan contenant une action abstraite non instanciée et deux liens ouverts.

Cette action abstraite a été introduite pour résoudre un lien ouvert : on a donc ajouté le lien causal nécessaire. Mais sa précondition n'est pas assurée : son introduction a aussi créé un nouveau lien ouvert. Un lien causal avec l'action initiale suffit à résoudre ce défaut. On obtient alors le plan suivant :



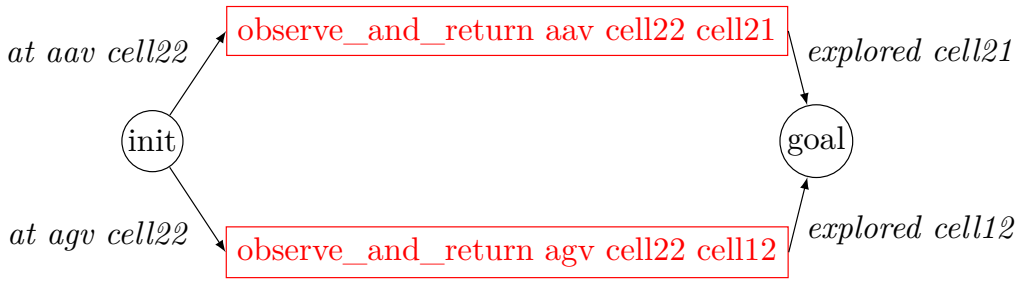
(d) Plan contenant une action abstraite non instanciée et un lien ouvert.

De même que précédemment, pour résoudre le deuxième lien ouvert on ajoute une action abstraite.



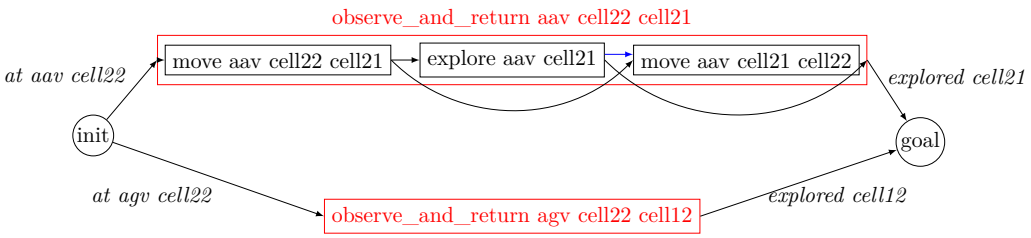
(e) Plan contenant deux actions abstraites non instanciées et un lien ouvert.

Dans ce cas de figure, les actions abstraites ne partagent pas de conflits. Elles peuvent donc être concurrentes dans le plan. Si elles avaient concernées le même robot, les conflits se seraient chevauchés et une menace aurait été introduite. De même que précédemment, le dernier lien ouvert peut se résoudre facilement :



(f) Plan abstrait contenant deux actions abstraites non instanciées.

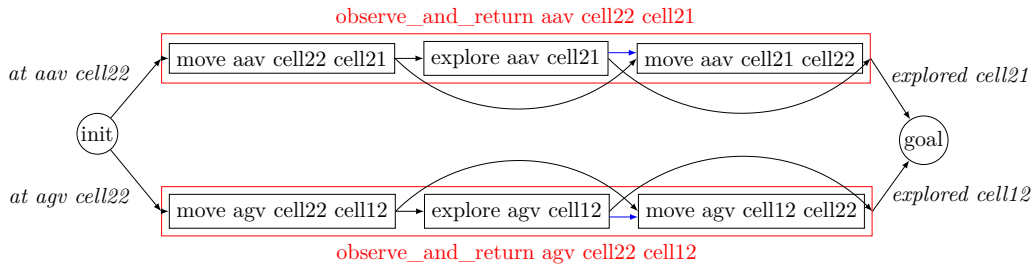
On a alors un plan abstrait : ses seuls défauts sont la présence de deux actions abstraites non instanciées. Pour résoudre un de ces défauts, il faut instancier une action en utilisant une de ses méthodes. La seule méthode disponible est alors utilisée :



(g) Plan abstrait contenant une action abstraite non instanciée et une action abstraite instanciée. Pour ne pas surcharger la figure, les tâches initiales et finales des méthodes n'ont pas été représentées.

Pour ne pas surcharger la figure, les fluents associés à chaque lien causal n'ont pas été représentés. Cette méthode est complète et son introduction n'a pas généré de nouveaux défauts. Il n'en reste alors plus qu'un seul : la présence d'une action

abstraite non instanciée. Après résolution de ce défaut, on obtient le plan présenté figure 3.3h.



(h) Plan solution du problème. Pour ne pas surcharger la figure, les tâches initiales et finales des méthodes n'ont pas été représentées.

Ce plan est valide car il ne présente aucun défaut. Il ne résout pas le problème de manière optimale (car le retour des robots à leur position initiale n'est pas nécessaire) mais il fournit une solution qui se conforme à ce que l'utilisateur a décrit avec les actions hiérarchiques.

Préconditions statiques

Les préconditions additionnelles d'une méthode (*i.e.* celles qui ne sont pas aussi des préconditions de l'action abstraite associée) peuvent être *statiques* ou non. Les préconditions statiques sont des contraintes qui peuvent être résolues durant le calcul des actions disponibles et qui n'ont pas besoin de l'état courant pour être déterminées. Ces contraintes peuvent donc être utilisées lors de l'instanciation du problème (*i.e.* la création de toutes les actions disponibles en utilisant les actions du domaine PDDL et la liste des objets du problème PDDL). Ces préconditions peuvent être utilisées dans des méthodes complètes. Pour les préconditions non statiques, il est nécessaire de les intégrer à la tâche `:init` de chaque méthode. Lors de l'utilisation de cette méthode, un lien ouvert sera donc créé pour garantir que cette condition est validée lors de son exécution dans le plan. Ces méthodes sont donc nécessairement non-complètes.

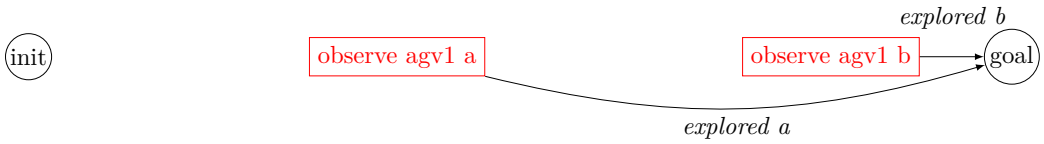
Exemple : Exemple de préconditions dans les méthodes

L'exemple des actions *observe_and_return* présentées page 63 utilise une précondition statique : l'égalité de 2 fluents ne dépend pas du plan courant. Ainsi, lors de la phase initiale d'instanciation du problème, une seule méthode est gardée par action sur les deux qui sont décrites.

Au contraire, la précondition `at ?r ?from` est non statique. Elle aurait pu être utilisée comme une précondition additionnelle des méthodes et non pas comme précondition de l'action abstraite. L'action hiérarchique n'aurait alors pas eu `at ?r ?from` comme précondition et la méthode n'aurait pas contenu d'actions de déplacement. Dans ce cas les actions hiérarchiques auraient pu être introduites sans précondition par le planificateur. Un plan abstrait aurait été plus rapide à être trouvé (car il y

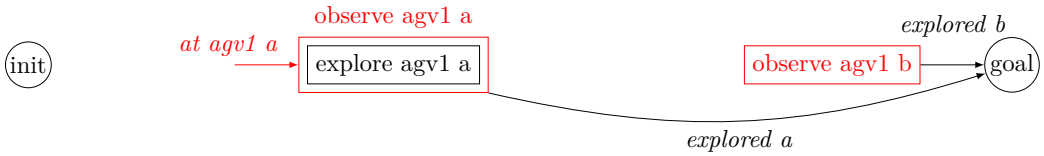
aurait eu moins de contraintes à satisfaire), mais à chaque instanciation, il aurait fallu ajouter les actions de déplacement correspondantes. Ce comportement est mis en évidence par la figure 3.4. Les actions abstraites utilisées ne comportent qu'une unique méthode, elle-même composée d'une seule tâche d'exploration. De plus, cette méthode a comme précondition `at ?r ?from`.
Le plan abstrait trouvé, pour un problème consistant à explorer deux lieux avec un unique robot, est alors :

Figure 3.4 – Exemple d'utilisation de préconditions non statiques dans une méthode.



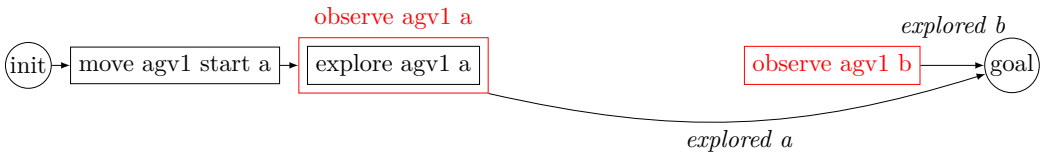
(a) Plan abstrait contenant deux défauts abstraits.

Comme les préconditions sur la position du robot n'apparaissent pas dans les préconditions de l'action hiérarchique, le planificateur ne peut pas raisonner sur la géométrie du problème. Par exemple, il ne peut pas raisonner sur le meilleur ordre des explorations en fonction des durées de déplacement, mais les conflits permettent de séparer temporellement les deux actions. Après l'instanciation d'une méthode, un lien ouvert apparaît :



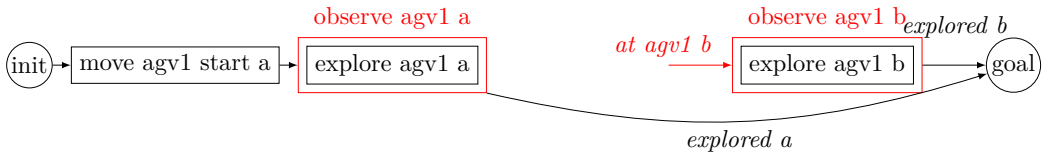
(b) Plan obtenu après instanciation d'une action abstraite. Un lien ouvert est apparu.

On peut le résoudre en ajoutant une action de déplacement :



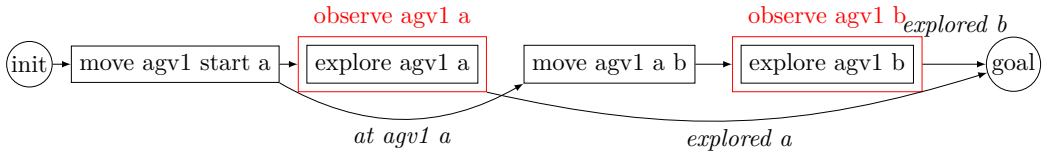
(c) Plan abstrait contenant un unique défaut abstrait.

Il ne reste alors dans le plan qu'un unique défaut abstrait. L'instanciation de la deuxième action donne alors :



(d) Plan obtenu après instanciation de la dernière action abstraite. Un nouveau lien ouvert est apparu.

Introduire la dernière action de déplacement et résoudre ses préconditions permet de trouver un plan solution.



(e) Plan solution.

Cette méthode permet d'éviter le raisonnement géométrique en forçant le planificateur à se tenir à un plan abstrait facilement calculable. Néanmoins, à chaque instanciation une recherche est nécessaire pour corriger les liens ouverts qui apparaissent.

Effets secondaires

Un autre ajout qui a été fait aux actions est la possibilité de définir des *effets secondaires*. L'idée est que, surtout pour les actions abstraites, certains effets correspondent aux buts réels des actions alors que les autres ne sont que des conséquences. Par exemple, si une action de livraison déplace un chargement et un véhicule de livraison, le but de l'action est le déplacement du paquet. Introduire une livraison dans le plan uniquement pour déplacer un véhicule n'est pas souhaitable. Plus généralement, une action ne devrait être ajoutée dans le plan que pour ses effets principaux, les effets secondaires n'étant utilisés que pour établir des liens causaux avec des actions qui sont déjà dans le plan.

Nous avons donc proposé une extension de PDDL où les effets de chaque action sont étiquetés comme étant principaux (par défaut) ou secondaires. Quand le planificateur résout un lien ouvert, il doit ajouter un lien causal vers une action déjà existante ou une nouvelle action. Dans le cas où il doit ajouter une nouvelle action dans le plan, il ne s'autorise à le faire que si le fluent fait partie de ses effets principaux.

Exemple : Effets secondaires d'une action

Considérons l'action `observe_and_return` de l'exemple de la page 63. Cette action a deux effets : l'observation d'une cellule et la présence du robot au point de départ. L'idée derrière cette action est évidemment d'observer une cellule, la position finale du robot n'est qu'un effet secondaire qu'il est nécessaire de mentionner pour la

cohérence du plan.
Par exemple si une autre action nécessite la présence d'un robot dans une cellule précisée, introduire une action d'observation de n'importe quelle autre cellule ne parait pas pertinent : une action de déplacement semble plus indiquée.
Ainsi dans la définition de l'exemple page 63, les effets peuvent être déclarés comme suit (en remplacement de la ligne 5)

```
1   :effect (explored ?to)
2   :side-effect (at ?r ?from)
```

Cette définition permet de réduire le nombre de plans résolvant un défaut de lien ouvert (en particulier pour la position des robots) donc de limiter l'arbre de recherche.

L'utilisation des effets secondaires revient à élaguer prématurément l'arbre de recherche. Cela peut donc conduire à se couper de meilleures solutions (en imposant des actions de déplacement supplémentaires non nécessaires par exemple) voire même d'empêcher de trouver des solutions (si aucune alternative n'existe pour établir un lien causal en utilisant les effets principaux d'une action).

3.2.3 Identification d'heuristiques de sélection des plans et adaptation à notre implémentation

Les performances des algorithmes de planification sont souvent fortement liées aux heuristiques qu'ils utilisent. VHPOP [Younes et Simmons 2003] a proposé une évaluation de différentes heuristiques pour un planificateur POP. Parmi ces heuristiques, nous avons identifié les heuristiques les plus prometteuses et les avons adapté à notre cas, en particulier à la gestion des actions hiérarchiques.

Deux heuristiques distinctes sont utilisées en POP : une heuristique pour classer les plans et une heuristique pour classer les défauts. Cette section étudie le premier type d'heuristique : le choix d'un plan à étendre.

Utilisation de h^{add}

Pendant la recherche, l'algorithme maintient en permanence une liste de plans partiels à explorer, comme pour une recherche utilisant A^* [Hart, Nilsson et Raphael 1968]. Et comme pour une recherche utilisant A^*, il est nécessaire d'ordonner ces éléments pour se concentrer sur les plans partiels les plus prometteurs. Plus spécifiquement, les plans sont évalués par $f(\Pi) = g(\Pi) + h(\Pi)$ où $g(\Pi)$ est le coût du plan partiel et $h(\Pi)$ une estimation du coût nécessaire à ajouter au plan pour passer de Π à un plan solution.

Dans le cadre d'une planification non temporelle, le coût d'un plan est son nombre d'actions. Plusieurs fonctions h ont été proposées : compter le nombre de défaut, compter juste le nombre de liens ouverts ou évaluer le nombre d'actions nécessaires pour chaque lien ouvert.

C'est cette dernière option qui est choisie pour VHPOP après comparaison avec les autres approches, et c'est celle que nous utiliseront. L'hypothèse simplificatrice permettant de calculer rapidement l'heuristique est celle de l'indépendance des liens causaux : on suppose qu'on peut évaluer séparément le coût de chaque fluent à établir et faire la somme de ces coûts pour obtenir le nombre d'actions à ajouter dans le plan. C'est souvent une sur-estimation du nombre d'actions nécessaire car une action peut contribuer à la réalisation

de plusieurs buts, mais dans certains cas cela peut aussi être une sous-évaluation si deux objectifs sont contradictoires l'un avec l'autre. C'est donc l'heuristique h^{add} présenté à la définition 23 page 31 qui est utilisée pour évaluer le coût de chaque lien ouvert d'un plan.

Définition 38 : h^{add} (POP)

Soit Π un plan partiel et $OL(\Pi)$ la liste des liens ouverts de Π. Chaque lien ouvert est définit par un instant temporel t et un fluent f.

$$h^{add}(\Pi) = \sum_{(t,f)\in OL(\Pi)} h^{add}(f)$$

Cette heuristique n'utilise que l'état initial et le fluent évalué, on peut donc la calculer une fois pour toute en début de recherche et pour tous les fluents. Un de ses inconvénients est aussi de n'utiliser que l'état initial et pas le plan courant pour évaluer un lien ouvert. C'est sur ce constat que VHPOP en a proposé une modification appelée *reuse* qui consiste à ne pas prendre en compte un lien ouvert dans le calcul de h^{add} si le fluent associé est présent dans les effets d'une action précédente. Dans ce cas, la contrainte de précédence sera ajoutée quand le lien causal sera inséré : il suffit juste que cette précédence soit possible pour ignorer le coût d'un lien ouvert. Cela revient à dire que si un fluent a déjà été établi dans le plan, le coût de le ré-établir est nul car il pourra être réutilisé.

Définition 39 : h^{add}_{reuse}

Soit Π un plan partiel et $OL(\Pi)$ la liste des liens ouverts de Π.

$$h^{add}_{reuse}(\Pi) = \sum_{(t,f)\in OL(\Pi)} \begin{cases} 0 & \text{si } \exists \tau \in \mathcal{T}(\Pi) \mid f \in Add(\tau) \wedge (\tau \prec t \text{ est possible}) \\ h^{add}(f) & \text{sinon} \end{cases}$$

Cette modification est la bienvenue, mais dans certains cas elle n'est effectivement pas suffisante. En effet, l'établissement d'un fluent ne signifie pas qu'il sera toujours disponible. Nous avons donc proposé une extension de cette heuristique, appelée *advance reuse* (ou *areuse*) qui consiste à ne pas prendre en compte un lien ouvert si le fluent est établi par une action précédente et n'est effacé par aucune action qui se situe entre cette date d'établissement et la date du lien ouvert. Dans ce cas, la précédence de l'action destructrice n'est prise en compte que si elle est obligatoire dans ce plan, sinon l'ajout du lien causal et la résolution de la menace qui en résulte forment un plan valide résolvant le lien ouvert. Plus formellement, on a :

Définition 40 : h^{add}_{areuse}

Soit Π un plan partiel et $OL(\Pi)$ la liste des liens ouverts de Π.

$$h^{add}_{areuse}(\Pi) = \sum_{(t,f)\in OL(\Pi)} \begin{cases} 0 & \text{si } \exists \tau \in \mathcal{T}(\Pi) \mid f \in Add(\tau) \wedge (\tau \prec t \text{ est possible}) \wedge \\ & \forall \tau_1 \in \mathcal{T}(\Pi) \mid f \in Del(\tau_1) \implies (\tau_1 \prec \tau \vee \tau \prec \tau_1) \\ h^{add}(l) & \text{sinon} \end{cases}$$

Exemple : Comparaison entre h^{add}, h^{add}_{reuse} et h^{add}_{areuse}

Considérons un exemple simple où le but du problème est de planifier des observations pour un satellite. Avant chaque observation, l'instrument utilisé doit être calibré d'une manière précise, les différentes observations pouvant ou non utiliser la même calibration.
Le plan partiel contient une calibration de type 1 (t_1), une observation puis une nouvelle calibration de type 2 (t_2) pour une observation future. Une nouvelle observation est ajoutée au plan, nécessitant une calibration de type 1. Cet exemple est illustré à la figure 3.5.

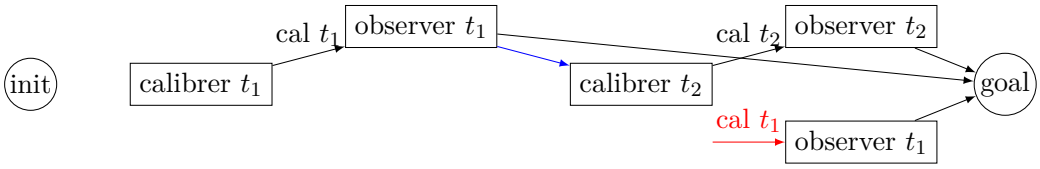


Figure 3.5 – Comparaison entre h^{add}, h^{add}_{reuse} et h^{add}_{areuse}. Les liens causaux sont en noir, les liens temporels en bleu et les liens ouverts en rouge.

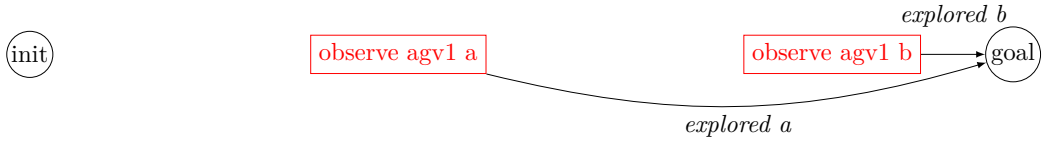
En utilisant h^{add}, le coût du seul lien causal du plan est égal au coût d'établissement de *cal* t_1 à partir de l'état initial. Donc $h(\Pi) = 1$.
En utilisant h^{add}_{reuse}, la nouvelle action n'étant pas contrainte temporellement, elle peut avoir lieu après *calibrer* t_1. Son coût est donc nul. Donc $h(\Pi) = 0$.
En utilisant h^{add}_{areuse}, on prend aussi en compte l'action *calibrer* t_2. Si la nouvelle action n'a pas de contraintes temporelles, alors il sera possible de la contraindre avant *calibrer* t_2. Dans ce cas, le coût du lien ouvert est nul : $h(\Pi) = 0$. Mais si la nouvelle action doit être effectuée après *calibrer* t_2 (par une contrainte qui n'est pas représentée ici), alors son coût heuristique sera le même que si la première calibration n'avait pas eu lieu auparavant : $h(\Pi) = 1$.

L'interaction entre les heuristiques et les actions hiérarchiques doit aussi être étudiée. A première vue, et dans le cas où les méthodes des actions hiérarchiques sont complètes, ces heuristiques sont directement transposables. Néanmoins un problème se pose lors de l'instanciation d'une action abstraite : le nombre d'actions augmente instantanément. Lors d'une exploration classique, si un raffinement fait augmenter le coût du plan sans changer l'estimation heuristique alors tous les plans résultants seront considérés comme étant beaucoup moins bon. Imaginons un cas où une multitude d'alternatives existe pour résoudre un problème. Si l'ordre de résolution des défauts importe peu, plusieurs plans abstraits peuvent être présents dans la frontière avec chacun une estimation heuristique proche. Maintenant si le « meilleur » est étendu et que son coût augmente, il sera considéré comme étant moins bon que toutes les alternatives car l'heuristique n'a pas pu anticiper cette augmentation. Toutes les alternatives devront donc être explorées avant que le planificateur considère à nouveau le meilleur plan abstrait qu'il avait jusque-là.

Exemple : Nécessité d'adaptation des heuristiques à l'instanciation d'une action

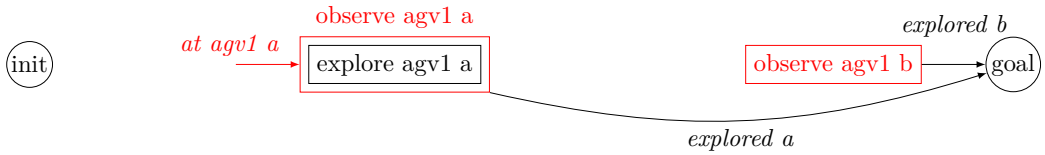
En reprenant l'exemple de la page 70, on peut évaluer le coût heuristique du premier plan abstrait obtenu (rappelé figure 3.6).

Figure 3.6 – Exemple présentant une augmentation du coût heuristique d'une plan lors de l'instanciation d'une action abstraite.



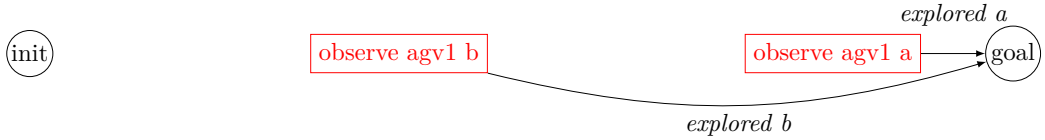
(a) Plan abstrait contenant deux défauts abstraits.

Il y a deux actions dans le plan, et aucun lien ouvert. On a donc $f = g+h = 2+0 = 2$. Une fois la première action instanciée, on a :



(b) Plan obtenu après instanciation d'une action abstraite. Un lien ouvert est apparu, ce qui fait augmenter le coût heuristique de h^{add}.

Il y a toujours deux actions dans le plan (on ne compte pas les actions instanciées), et un lien ouvert. Si ce lien ouvert n'est pas déjà établi dans l'état initial, il faut au moins une action pour l'établir : h^{add} vaut donc au moins 1. On a donc $f = g + h = 2 + 1 = 3$. On va donc explorer en priorité un autre plan abstrait avec $f = 2$ avant de continuer les instanciations. Par exemple le plan suivant, obtenu en changeant l'ordre d'exploration des cellules :



(c) Plan obtenu après instanciation d'une action abstraite. Un lien ouvert est apparu, ce qui fait augmenter le coût heuristique de h^{add}.

Une fois sa première instanciation faite, il aura un coût $f = 3$ et sera donc jugé de même qualité que l'autre plan instancié. Si on avait beaucoup de plan abstraits équivalents comme ces deux-là, il faudrait tous les instancier une fois avant de pouvoir continuer leur exploration.

De même, si l'action abstraite contenait plus qu'une action, g aurait augmenté ce

qui aurait fait augmenter f. Ainsi, l'instanciation fait augmenter f ce qui incite le planificateur à explorer les autres plans équivalents (avant instanciation) en priorité. Par exemple le plan inversant l'ordre de réalisation des actions hiérarchiques a aussi $f = 2$ et sera donc exploré en priorité.

Ce problème provient du fait que l'heuristique ne prend pas en compte les actions abstraites encore dans le plan. Si une action abstraite avait une seule méthode, alors on pourrait prendre en compte le nombre d'actions composant cette action dans le calcul de h^{add} et ce problème ne surviendrait pas. Mais dans le cas où plusieurs méthodes sont possibles, il faut soit adapter notre heuristique soit trouver un moyen de contourner le problème.

Nous avons adapté l'heuristique en prenant en compte l'augmentation minimale de la métrique. Si on s'intéresse au nombre d'actions, on calcule (récursivement) le nombre minimum d'actions élémentaires qu'une action abstraite va ajouter en fonction de ses méthodes. Ainsi, lors d'une instanciation, on limite au plus ce « saut » de la valeur heuristique. Voire, dans la plupart des cas rencontrés lors de ces travaux, on l'élimine car il n'y a soit qu'une unique méthode soit que des méthodes ayant le même coût.

Le même phénomène se produit quand une méthode non complète est utilisée : un ou plusieurs défauts sont introduits lors de l'instanciation. Ces défauts vont faire augmenter le coût heuristique, ce qui aura le même effet que celui d'augmenter directement le coût du plan.

Une méthode pour contourner ce problème peut être mise en place quand la description des actions hiérarchiques vérifie une propriété précise. En effet, si un plan abstrait mène toujours à une solution, alors on peut se contenter de chercher parmi les descendants du plan abstrait considéré, sans avoir à les comparer à des plans précédents.

Définition 41 : Domaine hiérarchiquement bien formé

Un domaine est dit *hiérarchiquement bien formé* si tout plan abstrait, considéré comme point de départ initial d'une recherche, mène à une solution *i.e.* l'algorithme 3 appliqué à tout plan abstrait retourne un plan valide.

Cette propriété ne nécessite pas que les méthodes soient complètes : il est possible de devoir faire une recherche pour trouver la bonne instanciation et pour résoudre tous ses défauts, y compris en ajoutant de nouvelles actions. A l'inverse, si un domaine ne possède que des méthodes complètes, qu'il n'y a pas de cycle dans la hiérarchie des actions et qu'il n'y a pas d'action possédant un effacement qui ne soit pas dans leur liste de préconditions, alors le domaine est hiérarchiquement bien formé. En effet, l'instanciation de chaque action ne peut qu'introduire de nouveaux défauts abstraits et l'absence de cycle garantit qu'au bout d'un certain nombre d'instanciations il n'y aura plus de défaut abstrait.

Si un domaine vérifie cette propriété, alors il est possible d'oublier (ou tout du moins de mettre de côté) tous les autres plans de la frontière dès qu'un plan abstrait est rencontré. Cela permet d'éviter d'explorer un palier complètement avant de continuer d'étendre le meilleur plan abstrait trouvé. Cette option a été ajoutée à la description du problème fournie par l'utilisateur. Si elle est présente, le planificateur oubliera tous les autres plans partiels dès qu'un plan contenant uniquement des défauts abstraits est rencontré. Ce comportement est analogue aux systèmes qui planifient à plusieurs niveaux de hiérarchie sans backtrack :

une fois que le plan à haut niveau est calculé, il est utilisé par le planificateur plus bas dans la hiérarchique comme un but.

Cette méthode a été utilisée dans les domaines qui présentent des actions abstraites non complètes mais n'a pas été nécessaire pour les domaines présentant des actions complètes car l'estimation du coût de l'action d'après ses méthodes se révèle juste.

Une heuristique pour départager les égalités

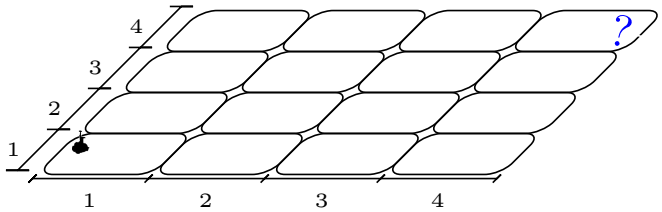
VHPOP propose une autre évaluation des plans, basée sur l'*effort*. L'effort est une estimation du nombre de plans qu'il sera nécessaire d'explorer avant d'arriver à une solution. Autrement dit, l'effort évalue le nombre de raffinements nécessaires pour atteindre une solution. Cela permet donc de se concentrer sur les plans proches de la solution, sans considérer leur qualité.

VHPOP propose d'utiliser cette heuristique pour départager les égalités entre évaluations heuristiques de la qualité des plans. Le calcul se fait de manière similaire à h^{add} pour évaluer l'effort nécessaire à la résolution d'un fluent donné (cf. définition 23 page 31) à la différence que l'effort d'un fluent présent dans l'état initial est de 1 et le coût d'une action est de 1. Dans les deux cas, il faut un raffinement pour régler un lien ouvert par l'ajout d'un lien causal (avec ou sans introduction d'une nouvelle action).

Exemple : Intérêt de l'effort dans l'heuristique de plans

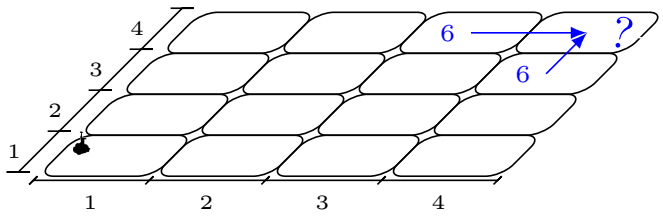
Supposons un problème de déplacement d'un robot sur une grille où les seuls déplacements autorisés sont les déplacements entre cellules adjacentes. Le robot est dans la cellule $(0, 0)$ et le but est d'être dans la cellule $(3, 3)$. La frontière est triée selon $f(n) = g(n) + h(n)$ pour effectuer une recherche A^*, comme présenté précédemment page 73. Pour départager les égalités, on utilise l'estimation de l'effort. En cas d'égalité, on choisit au hasard le prochain plan exploré.

Figure 3.7 – Exemple de l'intérêt de l'utilisation de l'effort d'un plan.



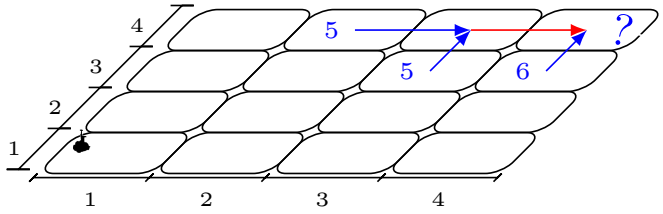
(a) Problème initial.

On a un unique lien ouvert sur la position du robot en $(4, 4)$ (pour satisfaire le but), avec un coût estimé de $h = 6$ et un effort estimé de 7. La première expansion consiste à ajouter une action qui mène le robot à son but. Il y a deux solutions pour cela.



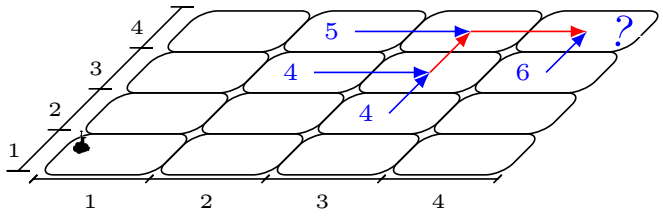
(b) Première expansion : 2 solutions pour atteindre la case $(4, 4)$. L'effort estimé est indiqué en bleu.

On obtient donc deux plans de coût $g = 1$, de coût heuristique restant $h = 5$ et d'effort 6. Ces deux plans sont équivalents : $f = 6$. Supposons que le premier plan exploré soit celui partant de $(3, 4)$.



(c) Deuxième expansion : 2 solutions pour atteindre la case $(3, 4)$ avec un coût minimal et le plan partiel issu de l'expansion précédente. L'effort estimé est indiqué en bleu.

Dans ce cas on a 4 plans dans la frontière. Parmi les 3 nouveaux, un plan consiste à partir de la cellule $(4, 4)$. Ce plan a une taille de 2 et un coût heuristique de 6 : on a donc $f = 8$. Son coût est plus élevé que les autres, il ne sera donc pas exploré en premier (il n'est pas représenté sur la figure). Les deux autres ont une taille de 2 et un coût heuristique de 4 : on a donc $f = 6$. Ils ne sont donc pas distingués du plan issu de la première expansion (partant de la case (4,3)) par le coût. Par contre, ils ont un effort estimé de 5. Ils seront donc explorés en priorité. Supposons qu'un de ces deux plans soit choisi.



(d) Troisième expansion : 2 solutions pour atteindre la case $(3, 3)$ avec un coût minimal. L'effort estimé est indiqué en bleu.

Là encore, un certain nombre de plans dans la frontière ont le même coût heuristique. Mais l'utilisation de l'effort permet de se concentrer sur les plans les plus proches d'une solution, au détriment des plans les plus anciens. Ici, cela permettra d'atteindre un plan solution en un nombre minimal d'expansions (7) et sans avoir à explorer toute la grille.

Nous avons ajouté lors de l'évaluation d'un plan l'effort nécessaire pour résoudre les menaces et les défauts abstraits. Dans les deux cas, il suffit d'un raffinement pour les régler : l'ajout d'une contrainte temporelle ou une instanciation.

En cas d'égalité persistante après l'utilisation de l'effort, il est aussi possible de trier les défauts en LIFO. Cela permet de concentrer la recherche sur les derniers plans ajoutés.

Heuristiques de planification temporelle

Dans le cas de la planification temporelle, le but est non pas d'optimiser le nombre d'actions mais la durée finale du plan. Dans un cadre multirobot, ces objectifs peuvent être extrêmement différents car plusieurs robots agissant en parallèle vont faire augmenter le nombre d'actions sans faire changer la durée du plan. Il est alors souhaitable de décharger le robot le plus lent de certaines actions, même si cela en rajoute plus à un robot dont le plan serait moins rempli.

Le calcul de h^{add} pour un fluent donné doit alors être adapté pour ne pas prendre en compte le nombre d'actions mais bien la durée des actions. Dans ce cas, l'heuristique h^{add} fait une approximation bien plus forte que dans le cas séquentiel. En effet, deux buts distincts peuvent très souvent se faire en parallèle dans un cadre multirobot et l'ajout du temps nécessaire pour faire chacune d'elles est une très forte approximation.

Par contre, l'intégration du coût d'une action abstraite est plus naturelle. Comme toute action élémentaire, une action abstraite non instanciée est aussi représentée par deux instants temporels : son début et sa fin. Ses deux instants doivent être séparés au moins de la durée de l'action. Dans le cas d'une action abstraite, pour ne pas sur-contraindre le problème, nous avons choisi d'utiliser la durée minimale d'une méthode comme durée de l'action abstraite correspondante. Bien que cela puisse sous-évaluer la durée réelle (comme dans le cas de la planification séquentielle), la durée totale du plan sera calculée sans faire l'hypothèse que ces coût sont additifs. Si plusieurs actions abstraites sont en parallèle dans le plan, la durée estimée du plan sera bien le maximum des durées des actions et non pas la somme de ces durées.

3.2.4 Identification d'heuristiques de sélection des défauts et adaptation à notre implémentation

Étant donné un plan, cette heuristique doit choisir le prochain défaut à résoudre. Différents critères ont été proposés dans la littérature : le type de défaut, le nombre de choix qui s'offrent pour résoudre ce défaut, l'ordre de création des défauts ou une évaluation heuristique du coût d'établissement d'un fluent.

Lorsque le type de défaut est pris en compte dans VHPOP, c'est majoritairement pour résoudre les menaces avant les liens causaux. De plus, quand les menaces et les liens ouverts sont traités de la même manière, ils sont ordonnés en commençant par les défauts présentant le moins de résolveurs possibles. Or les menaces étant limitées à 2 résolveurs maximum dans notre cas (où les actions sont complètement instanciées), ils se retrouveraient très souvent à être résolus parmi les premiers. Toutes les heuristiques intéressantes pour VHPOP traitant donc les menaces en premier, nous avons choisi de les résoudre avant les liens ouverts.

Dans notre cas, il a aussi fallu choisir dans quel ordre résoudre les défauts abstraits. L'ordre le plus naturel semblait être de les résoudre en dernier. En effet, cela permet de calculer d'abord un plan valide comprenant des actions abstraites (*i.e.* un plan abstrait) avant de raffiner ce plan en instanciant ses actions. C'est le même principe qui est utilisé dans les architectures de planification à plusieurs niveaux : on calcule d'abord un plan valide

à haut niveau avant d'essayer de raffiner à un niveau inférieur. De plus, cela permet de faire une plus grande partie de la recherche avec un plan comprenant un faible nombre d'actions ce qui accélère la recherche. Dernier avantage : si les méthodes des actions hiérarchiques sont complètes ou au moins que le domaine est hiérarchiquement bien formé, alors une fois qu'il ne reste que des défauts abstraits on peut oublier tous les autres plans dans la frontière car on est sûr de trouver une solution avec le premier rencontré.

Les menaces sont ordonnées par nombre de solutions : nous avons décidé de résoudre d'abord les menaces possédant le moins de résolveurs. Cela revient à éliminer en premier les plans impossibles temporellement (0 résolveur) puis à ajouter les contraintes imposées avant de résoudre les autres menaces. Les tests ont montré que finalement cet ordre de résolution ne faisait pas une grosse différence : quelque soit l'ordre utilisé, la résolution de toutes les menaces aboutira aux mêmes plans sans gros impact sur le temps de calcul.

Concernant les liens ouverts, nous avons testé l'ordre de VHPOP : l'utilisation du coût de chaque fluent et la résolution du plus coûteux en priorité. Une amélioration de ce classement considère en priorité, avant la prise en compte du coût, les liens ouverts qui ont été ajoutés le plus récemment. L'idée est de faire une recherche plus locale : on se concentre sur la résolution d'un but de haut niveau avant de passer aux suivants. Cela permet de résoudre chaque but séquentiellement plutôt que d'essayer de mener toutes les recherches en parallèle. C'est cette solution que nous avons retenue.

Concernant les actions abstraites, notre choix a été guidé par un cas d'application particulier. Dans le cas où des méthodes complètes existent pour chaque action, l'ordre de résolution n'a pas d'importance car toutes les instanciations sont indépendantes. Mais dans certains cas, l'instanciation d'une action abstraite a besoin de l'instanciation des actions précédentes.

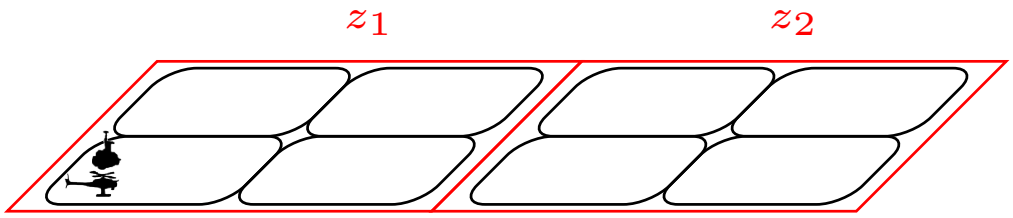
C'est le cas quand le domaine a été conçu pour planifier avec plusieurs niveaux de hiérarchie. Par exemple un domaine peut être écrit avec des actions hiérarchiques qui régissent les actions d'équipes de robots alors que les actions élémentaires concernent uniquement un robot seul. Lors de la recherche, on commence par raisonner sur les fluents de position d'une équipe donnée et on introduit des actions abstraites. Ces actions abstraites doivent indiquer dans leurs effets quelle est la position finale de l'équipe pour pouvoir enchaîner avec une autre action abstraite. Mais elle peut éviter de définir la position de chaque robot. Elle peut par exemple posséder plusieurs méthodes, chacune faisant terminer les robots à des positions légèrement différentes sans que la position globale de l'équipe ne soit changée (cf. l'exemple suivant). Dans ce cas, lors de l'instanciation d'une action abstraite, des liens ouverts vont être créés concernant la position individuelle de chaque robot. Il est alors nécessaire de soit résoudre ces défauts après toutes les instanciations soit d'avoir accès aux positions des robots lors de l'instanciation. Pour éviter de forcer la première solution (afin de raisonner le plus longtemps possible avec des plans abstraits), nous avons privilégié la deuxième solution. Cela impose donc de résoudre les défauts abstraits dans l'ordre chronologique : c'est le choix qui a été fait.

Exemple : Planification hybride pour l'exploration multirobot

Supposons qu'une équipe de robots doive explorer un grand terrain. Il peut être intéressant de découper ce terrain en plusieurs zones pour raisonner d'abord sur le plan général d'exploration des zones avant de raisonner sur les mouvements

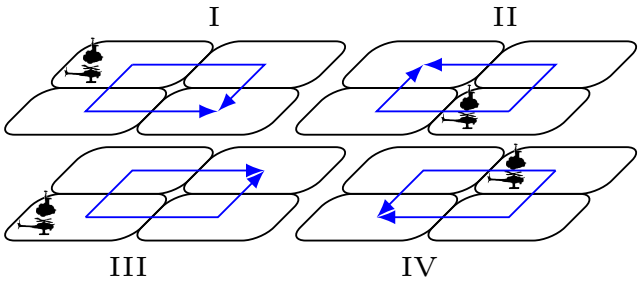
individuels des robots. De même il peut être intéressant de regrouper les robots en plusieurs sous-équipes pour leur confier des tâches globales avant de décider quelles actions élémentaires leur faire faire.
Supposons ici qu'une équipe composée de deux robots doivent explorer une grille de 8 cases. On sépare la grille en deux zones adjacentes :

Figure 3.8 – Exemple d'un problème d'exploration multirobot résolu en planification hybride.



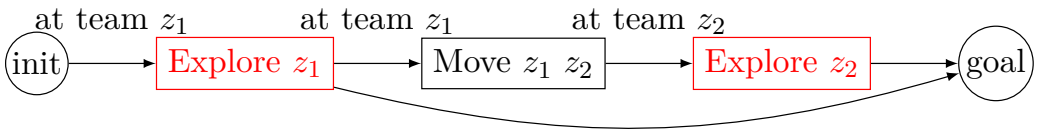
(a) Problème initial : séparation en zones.

Pour chaque zone, on introduit une action abstraite. Cette action produit comme effet l'observation de toutes les cellules de la zone. On définit quatre patrouilles différentes par zone, chacune avec un point de départ différent pour l'équipe (cf. figure 3.8b). La position initiale des robots est une précondition non statique de chaque méthode : les méthodes ne sont donc pas complètes.



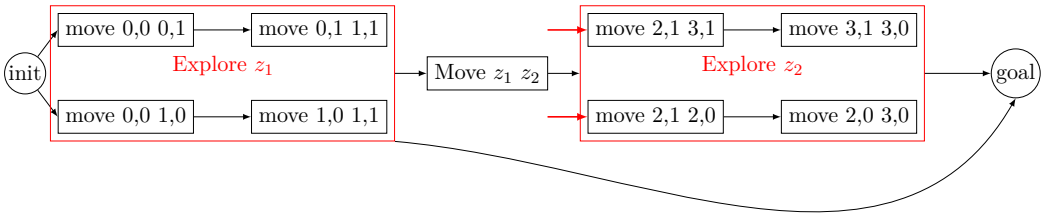
(b) Représentation des 4 patrouilles possibles utilisables pour l'exploration d'une zone.

De plus, pour assurer une cohérence temporelle aux plans abstraits, on introduit un fluent représentant la position de l'équipe. Ce fluent est introduit lors de la description du modèle pour guider le planificateur dans sa recherche. Il n'est pas lié à la position individuelle des robots et ne représente pas une position réelle, mais il est utile pour calculer des temps de déplacement cohérents. On introduit donc des actions élémentaires fictives agissant sur la position de l'équipe dont la durée est la durée minimale nécessaire pour un robot pour passer d'une zone à l'autre. Chaque action abstraite de patrouille nécessite que l'équipe soit dans la zone considérée. On peut alors obtenir le plan abstrait suivant :



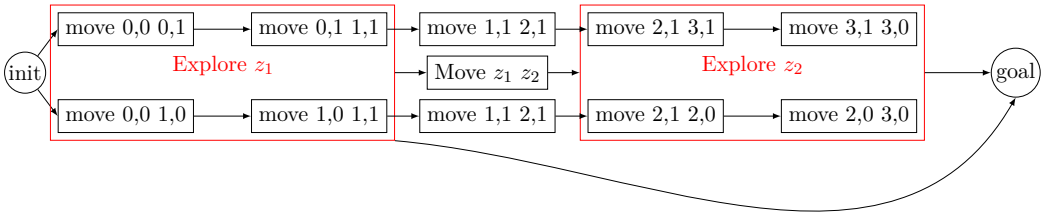
(c) Plan abstrait résolvant le problème.

On peut remarquer ici l'influence du fluent de position de l'équipe. Premièrement il permet de déterminer que le plan qui commence par explorer z_1 est plus efficace que celui commençant par z_2. En effet sans le fluent sur la position des équipes, les actions d'explorations n'auraient pas de précondition. Il n'y aurait pas de différence visible entre le plan explorant z_1 puis z_2 et celui explorant z_2 puis z_1.
De plus, il permet d'estimer plus finement la durée globale du plan : chaque action abstraite a la durée de sa plus petite méthode (ici 2) et l'action de déplacement d'équipe permet d'ajouter 1 à la durée du plan.
Une instanciation des méthodes donne alors :



(d) Plan complètement instancié mais non valide car présentant 2 liens ouverts.

Ici les méthodes ne sont pas complètes : leur introduction crée des liens ouverts. Pour simplifier la figure, on a considéré que les liens ouverts de la première méthode ont déjà été résolus en introduisant des liens causaux avec l'action initiale. Il reste alors ceux introduits par la deuxième méthode. Leur résolution nécessite l'introduction de nouvelles actions de déplacement, ce qui produit la solution suivante :



(e) Plan valide résolvant le problème.

Dans cet exemple, on voit bien l'intérêt de la planification à deux niveaux d'abstraction : on a commencé à chercher une solution en utilisant les fluents de position de l'équipe et les actions abstraites. Une fois le plan abstrait trouvé, on a pu le raffiner

pour en faire un plan solution. L'introduction de l'action de déplacement de l'équipe a permis d'affiner l'estimation de la durée du plan pour que le meilleur plan abstrait corresponde bien au meilleur plan solution.

3.2.5 Résumé des heuristiques utilisées dans HiPOP

h^{add} possède deux options (*reuse* et *areuse*) décrites aux définitions 39 et 40. h^{add} est utilisée à la fois par l'heuristique de sélection des plans et par l'heuristique de sélection des défauts.

Heuristique de sélection des plans L'heuristique de sélection des plans utilise l'algorithme A* : les plans sont triés en fonction de $f(\Pi) = g(\Pi) + h(\Pi)$. g compte le nombre d'actions élémentaires dans le plan. Pour évaluer un plan, h fait une somme du coût de chaque lien ouvert et de chaque défaut abstrait. Le coût d'un lien ouvert est calculé comme le coût h^{add} du fluent et le coût d'un défaut abstrait comme le nombre minimum d'actions élémentaire introduit par l'instanciation de l'action abstraite. En cas d'égalité, l'*effort* de chaque plan est utilisé. S'il y a encore égalité, on utilise un ordre LIFO.

L'option possible de cette heuristique est *time*. Si elle est présente, on cherche à optimiser la durée totale du plan solution. g calcule alors la durée du plan actuel et h n'évalue pas les défauts abstraits (leur durée est déjà prise en compte dans la durée globale du plan). h^{add} utilise la durée des actions comme coût (cf. définition 23 page 31).

Heuristique de sélection des défauts Les défauts sont triés par type. Les menaces sont résolues en priorité, puis les liens ouverts et finalement les défauts abstraits.

Les liens ouverts peuvent être triés de plusieurs manières :

- *threatFirst* : ordre LIFO, par défaut.
- *sorted* : trié par coût h^{add}, les plus coûteux d'abord.
- *local* : résout en premier les liens ouverts de l'action ajoutée le plus tard lors de la recherche et possédant des liens ouverts. Les liens ouverts de la même action sont triés comme avec *sorted*.
- *earliest*, en plus d'une des options précédentes : lors du tri (par l'option *sorted* ou au sein d'une même action avec *local*), privilégie les liens causaux qui arrivent le plus tôt dans le plan.

Les défauts abstraits sont résolus par ordre chronologique dans le plan : on résout en priorité ceux dont les tâches arrivent plus tôt.

3.3 Validation expérimentale de l'algorithme de planification hybride

Tous les choix présentés dans ce chapitre ont mené à la création d'un algorithme de planification écrit en C++ et nommé HiPOP (pour Hierarchical Partial Order Planner).

Nous avons souhaité comparer les performances de cet algorithme à d'autres algorithmes de planification pour valider les choix qui ont été faits et les performances de ce planificateur sur un ensemble de problèmes en vue de son utilisation dans un cadre multirobot.

3.3.1 Choix des domaines et définition des actions abstraites

Pour évaluer les performances de notre algorithme, il est nécessaire d'utiliser des domaines présentant des actions hiérarchiques. De tels domaines ne sont pas disponibles : le choix d'utilisation de PDDL pour avoir des planificateurs de référence nous limite à des domaines ne possédant pas d'actions abstraites.

Nous avons donc tenté d'identifier certains domaines provenant des IPC qui se prêteraient à une résolution hiérarchique et nous avons créé nos propres domaines représentant un problème d'exploration multirobot proche de celui que nous souhaitons résoudre. Des exemples de description des domaines et des problèmes sont disponibles à l'annexe B.

Domaines issus des IPC

Blocks Ce domaine consiste en un ensemble de blocs posés sur une table (cf. figure 3.9). Chaque bloc peut être posé sur la table ou sur un autre bloc. Les actions consistent à gérer une pince : elle peut prendre un bloc si aucun autre bloc n'est posé dessus et elle peut poser le bloc qu'elle tient sur n'importe quel autre bloc libre ou sur la table. Le but est d'obtenir un arrangement donné des blocs en pile.

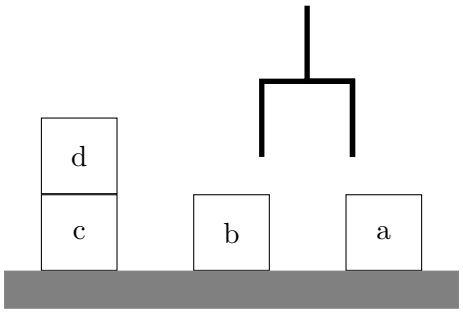


Figure 3.9 – Représentation du domaine *blocks* avec 4 blocs. La pince peut prendre n'importe quel bloc libre (ici a, b ou d) et le déposer sur un autre bloc libre ou sur la table.

Comme pour le domaine *satellite*, il est difficile de trouver des actions abstraites génériques qui agissent sur plusieurs blocs. Nous avons donc essayé de décrire les enchaînements d'actions qui arrivent le plus fréquemment : la prise et la dépose d'un bloc donné. Une action consiste à dégager un bloc en prenant celui qui se trouve au-dessus et en le plaçant sur la table. Une autre action consiste à mettre un bloc sur un autre si les deux sont libres. La définition du domaine et des actions nous a contraint à utiliser deux actions différentes selon que le bloc déplacé soit posé sur la table ou sur un autre bloc : ce ne sont pas les mêmes fluents qui sont dans les préconditions et dans les effets.

Les 30 problèmes utilisés sont ceux issus de l'IPC2.

Gripper Ce domaine est relativement simple : un robot possédant deux bras doit transporter des balles d'une salle à une autre salle adjacente (cf. figure 3.10). Les actions disponibles sont de prendre une balle avec un des bras, de se déplacer entre les deux salles et de déposer une balle présente dans un des bras. Bien que la solution soit évidente pour un humain, un planificateur peut être mis en difficulté par le nombre de choix qui s'offre à chaque étape. Même en considérant uniquement les plans de taille minimale (où le robot ne fait que des allers-retours avec 2 balles dans un sens et aucune dans l'autre), un problème

avec 10 balles possède 10! = 3628800 solutions différentes en fonction de l'ordre dans lequel les balles sont prises et avec quel bras.

Nous avons choisi ce problème car il se prête bien à une description par un ensemble d'actions hiérarchiques. En effet la forme des solutions recherchées est toujours la même : le robot doit prendre 2 balles dans la salle de départ, se déplacer, déposer les balles, revenir et refaire la même chose avec d'autres balles. Nous avons donc créé une action abstraite qui regroupe toutes ces actions en une et qui possède une unique méthode complète.

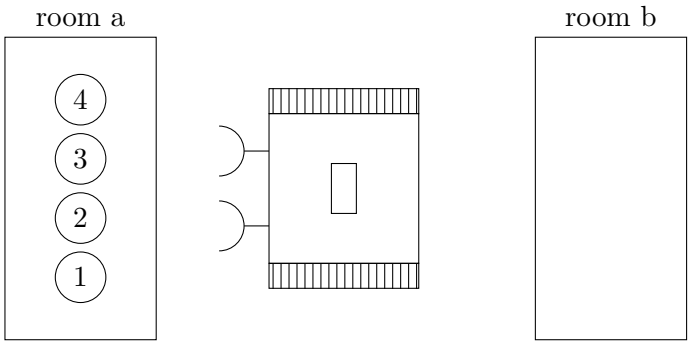


Figure 3.10 – Représentation du domaine *gripper* avec 4 balles. Le but est d'amener les 4 balles de gauche à droite, le robot ne pouvant en porter qu'une seule par bras à chaque fois.

Les 20 problèmes utilisés sont ceux issus de l'IPC1.

Logistics Ce domaine représente des problèmes de livraison de paquets entre villes. Des camions permettent de transporter des paquets entre les différents lieux d'une même ville et des avions permettent de le faire entre certains lieux donnés (les aéroports) appartenant à deux villes différentes (cf. figure 3.11).

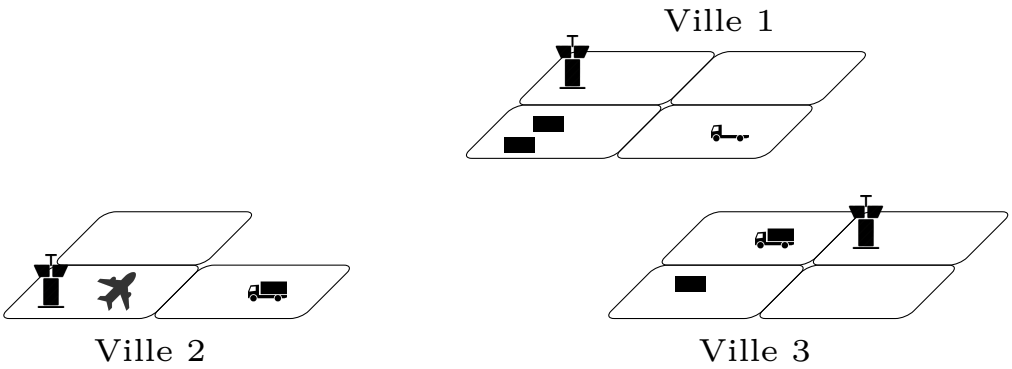


Figure 3.11 – Représentation du domaine *logistics* avec 3 villes. Chaque ville possède au moins un aéroport et un camion. Le but est d'amener chaque paquet à sa destination finale. Les camions peuvent se déplacer à l'intérieur d'une ville seulement alors que les avions ne voyagent qu'entre aéroports.

Dans le cas où chaque véhicule peut transporter plusieurs paquets simultanément, tous les objectifs sont liés et il est très difficile de faire des regroupements génériques d'actions

en actions hiérarchiques. Nous nous sommes intéressés au cas où chaque véhicule ne peut transporter qu'un seul paquet à la fois.

Dans ce cas, les méthodes concernent la livraison d'un paquet par un camion (s'il doit être livré dans la même ville), un avion (s'il doit être livré d'un aéroport à un autre) ou par une combinaison d'au maximum deux camions et un avion (s'il doit changer de ville). Les différentes actions correspondent aux différents cas de figure : livraison en passant par un aéroport, livraison dans la même ville, etc.

Les 48 problèmes utilisés ont été générés aléatoirement.

Satellite Ce domaine consiste à planifier des activités d'observation pour un satellite. Celui-ci possède plusieurs instruments de types éventuellement différents. Le but est de réaliser un certain nombre d'acquisitions, chacune nécessitant que le satellite soit tourné vers une direction donnée et utilise un type d'instrument donné.

Les acquisitions étant indépendantes mais nécessitant toutes une orientation particulière du satellite, il n'est pas aisé de rassembler plusieurs acquisitions en une unique action. Les actions abstraites utilisées combinent donc les actions qui viennent généralement successivement dans le plan : les actions pour calibrer un instrument (un changement d'orientation, l'allumage de l'instrument et sa calibration) et celles pour réaliser une acquisition (un changement d'orientation et une acquisition).

Les 20 problèmes utilisés sont ceux issus de l'IPC3.

Exploration multirobot

En plus des domaines issus des IPC, nous avons créé un domaine d'exploration multirobot similaire aux exemples présentés jusqu'ici, en particulier celui de la page 81, nommé *survivors*. L'idée principale était de disposer d'un ensemble de problèmes mettant en avant l'aspect multirobot (donc se prêtant à une modélisation hiérarchique des actions) et pouvant être générés de manière procédurale.

Le monde est représenté comme une grille découpée en zones, chaque zone étant composée de cellules. Des robots sont disponibles, arrangés en équipes. Chaque cellule doit être observée par un robot (qui doit y être présent pour l'observer) et un certain nombre de cellules contiennent de plus un hôpital. Des blessés sont répartis parmi les cellules et doivent être amenés à un hôpital pour y être hospitalisés. Un robot est nécessaire pour stabiliser un blessé et deux robots sont nécessaires pour le déplacer une fois stabilisé.

Les problèmes sont générés de manière procédurale en faisant varier le nombre de survivants, la taille de la grille, le nombre de zones, le nombre de robots, etc. De plus, la position des blessés et des hôpitaux est générée aléatoirement.

Les actions élémentaires disponibles sont les déplacements individuels des robots (la durée étant la distance de Manhattan, *i.e.* la distance nécessaire si on se déplace d'abord horizontalement puis verticalement indépendamment), les actions d'observation de la cellule courante d'un robot, les actions de stabilisation, d'hospitalisation et de déplacement d'un blessé.

Les actions abstraites correspondent à l'exploration d'une zone par une équipe de robots et au traitement d'un blessé (stabilisation, transport et hospitalisation) par une équipe. La méthode d'exploration d'une zone par une équipe est générée automatiquement en allouant un ensemble de cellules adjacentes à chaque robot. Les positions initiales des robots, n'apparaissant pas dans les méthodes, sont choisies pour minimiser la durée de

la patrouille. Les cellules sont réparties équitablement entre chaque robot de l'équipe et chaque robot explore un ensemble de cases adjacentes. Un exemple est donné ci-dessous.

Exemple : Patrouille dans le domaine *survivors*

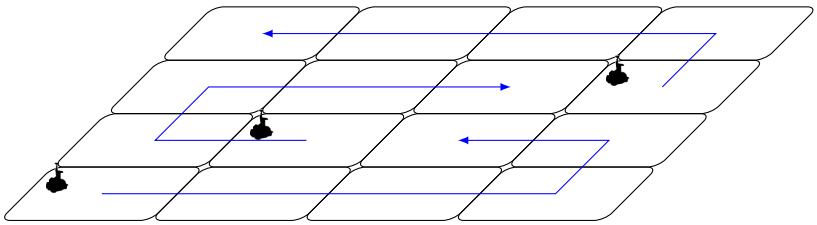


Figure 3.12 – Exemple d'une patrouille à 3 robots pour une zone de taille 4x4.

Les 72 problèmes utilisés comprennent de 2 à 3 équipes de 2 robots. Les zones sont des carrés ayant 2 à 4 cellules de coté. Il y a entre 4 et 8 zones. Chaque problème comprend 2 hôpitaux et de 2 à 5 blessés. Le problème le plus difficile concerne donc 6 robots sur une zone de $8 * 16 = 128$ cellules. Les plans solutions contiennent alors plus de 250 actions.

3.3.2 Protocole expérimental

Pour chaque domaine présenté, un ensemble de problèmes a donc été choisi (soit extrait des IPC, soit généré aléatoirement). Pour chacun de ces problèmes, un ensemble d'algorithmes de planification a été testé. Chaque test correspond donc à une configuration d'un planificateur donné utilisé sur un problème en particulier. Les résultats ont été regroupés en fonction des domaines.

Nous allons chercher à répondre à plusieurs questions :

- **Quels sont les meilleurs choix d'heuristiques ?** Pour faire de la planification temporelle, les choix les plus importants d'heuristiques semblent être ceux qui concernent le tri des défauts et l'option de *reuse*. On s'intéressera alors aux options suivantes :
  - Pour h^{add} : normal, *reuse* (abrégé en R) ou *areuse* (AR).
  - Pour l'heuristique de sélection des défauts : *threatFirst*, *sorted* et *local*.
- **Quelle est l'influence de l'heuristique *earliest* ?**
- **Est-ce que l'option *time* permet d'optimiser efficacement la durée des plans trouvés ?**
- **Quel est l'apport de l'utilisation des actions hiérarchiques pour la planification ?** Pour cela, on utilisera l'option *Bare* qui signale qu'aucune action hiérarchique n'a été fournie au planificateur. HiPOP-Bare est alors équivalent à un algorithme POP.
- **Comment HiPOP se compare-t-il avec d'autres planificateurs ?** Aucun autre planificateur hybride pouvant accepter notre format d'entrée n'étant disponible, il n'a pas été possible de le comparer directement à un autre planificateur hybride. Nous avons donc décidé d'utiliser des planificateurs ayant concouru aux IPC et capables de raisonnement temporel :

– VHPOP (Versatile Heuristic Partial Order Planner)[Younes et Simmons 2003]
– TFD (Temporal Fast Downward) [Röger, Eyerich et Mattmüller 2008]
– YAHSP (Yet Another Heuristic Search Planner)[V. Vidal 2011]

3.3.3 Résultats

Quels sont les meilleurs choix d'heuristiques ? La figure 3.13 présente les résultats de différentes configurations temporelles sur les 5 domaines présentés précédemment pour plusieurs configurations de HiPOP. On s'intéresse ici au nombre de problèmes qui ont été résolus en un temps donné : chaque point sur la courbe indique donc combien de problèmes ont été résolus en moins d'un certain temps. Une courbe située au-dessus d'une autre indique donc un planificateur plus rapide, et une courbe qui termine plus haut qu'une autre indique un planificateur capable de résoudre plus de problèmes.

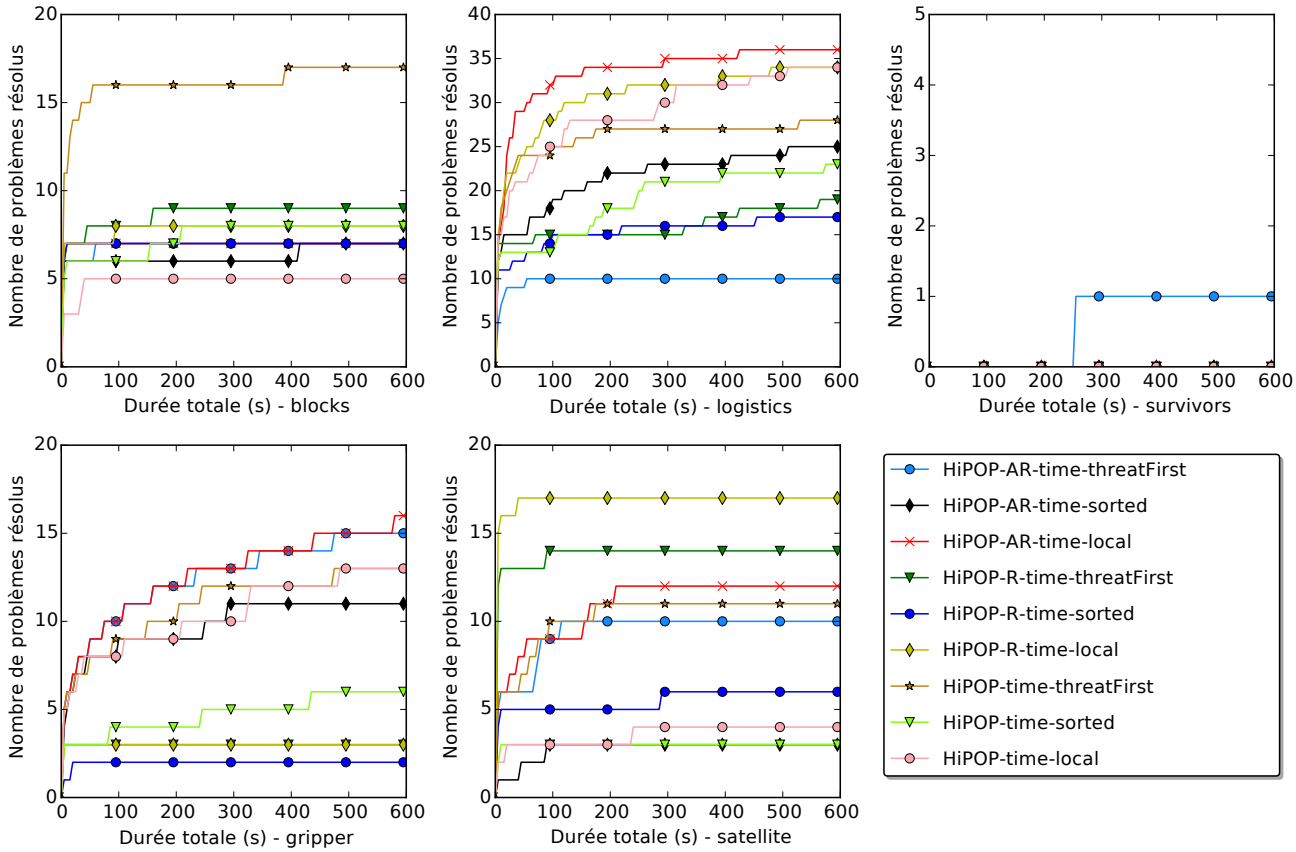


Figure 3.13 – Nombre de problèmes résolus par plusieurs configurations d'HiPOP sur différents domaines. Comparaison des utilisations de h^{add}, h^{add}_{reuse}, h^{add}_{areuse}, *sorted* et *local*. Chaque graphique correspond à un domaine différent et représente en ordonnée le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse.

On remarque premièrement que les résultats sont assez différents sur chaque domaine. Par exemple sur le domaine *blocks*, la meilleure configuration n'utilise ni *reuse* ni *areuse* contrairement aux autres domaines. Cela est sans doute dû à la description des buts. En effet, les buts sont décrits par un ensemble de fluents de type `on a b` indiquant que le bloc

`a` doit être sur le bloc `b`. Il est donc possible, si le but est d'avoir `a` sur `b` lui-même sur `c`, d'avoir `a` sur `b` alors que `c` est ailleurs. Le planificateur va alors croire qu'il a atteint la moitié des buts alors qu'en fait il en est loin : il est obligé de retirer `a` de `b` pour atteindre ses objectifs. Les buts sont donc fortement couplés et il n'est pas garanti d'être plus proche de la solution si plus de fluents sont garantis : les heuristiques faisant cette hypothèse sont donc pénalisées.

Dans le domaine *gripper*, le robot doit continuellement passer de la première salle à la deuxième salle. L'utilisation de l'heuristique *reuse* lui indique donc, après son premier aller-retour, que cela ne prendra sans doute pas de nouvelle action de se rendre dans l'autre salle. La durée de tous les déplacements suivants est donc négligée et cela perd le planificateur dans l'exploration de « plateaux » (*i.e.* un ensemble de plans partiels similaires partageant le même coût heuristique) avant de pouvoir trouver une solution. C'est ce qui explique que les configurations utilisant *reuse* soient les moins performantes. L'heuristique *areuse* montre ici tout son intérêt en détectant à juste titre que si le robot a fait un aller retour, cela lui coûtera quand même encore du temps s'il doit retourner dans la deuxième salle.

Dans le domaine *logistics*, l'influence de l'heuristique *local* semble importante : les trois meilleures configurations l'utilisent. Par contre une configuration qui marchait bien dans *gripper* (HiPOP-AR-time-threatFirst) ne parvient pas à résoudre plus de 10 problèmes ici. Dans le domaine *satellite*, l'heuristique *reuse* semble adaptée (les deux meilleures configurations l'utilisent) alors que l'heuristique *sorted* ne fonctionne pas du tout.

Le domaine *survivors* montre des résultats surprenants : aucune configuration de HiPOP ne résout plus d'un problème sur les 72 proposés. Pourtant les premiers cas ne sont pas compliqués. Pour essayer de comprendre ce qui se passe, on s'est intéressé au calcul des plans abstraits. La figure 3.14 présente le nombre de problèmes pour lesquels un plan abstrait a été trouvé et donne une explication à ce phénomène. En fait, certaines configurations trouvent un plan abstrait dans quasiment tous les cas et en moins d'une minute. Le problème est alors de raffiner ce plan abstrait en un plan valide. Une analyse plus poussée montre que le premier obstacle concerne les liens ouverts introduits lors de la résolution des défauts abstraits. En effet lorsqu'une action d'exploration d'une équipe doit être instanciée, des liens ouverts apparaissent sur la position individuelle des robots. Ces liens ouverts ne peuvent être résolus que par l'ajout d'actions de déplacement des robots. Ces actions ont aussi comme précondition la position des robots : il est nécessaire de pouvoir utiliser une position de départ pour résoudre ces liens ouverts. Or si l'action abstraite précédant celle qui doit être instanciée n'est pas instanciée, cette position de référence n'existe pas. Le planificateur ne peut donc pas établir ces liens ouverts et ne s'en rend même pas compte : il essaye continuellement d'ajouter des actions de déplacement sans jamais réduire le nombre de liens ouverts : pour chaque lien ouvert corrigé il en introduit un nouveau. Plus généralement, ce problème vient de notre volonté de planifier avec deux niveaux de hiérarchie : au niveau des équipes et au niveau des robots. Lors du passage au raisonnement sur les robots, certains fluents ne sont pas encore disponibles et le raisonnement ne peut donc pas avoir lieu.

Plusieurs solutions peuvent pallier ce problème. On pourrait décider d'instancier toutes les actions abstraites en même temps, ainsi il est garanti que tous les éléments nécessaires pour raisonner au niveau des robots sont présents. Mais cette solution introduit une combinatoire importante : au lieu de raisonner individuellement sur chaque action abstraite pour choisir la méthode, on doit toutes les choisir ensemble. Une autre solution consiste à garantir que les actions abstraites sont instanciées dans l'ordre chronologique du plan. Ainsi

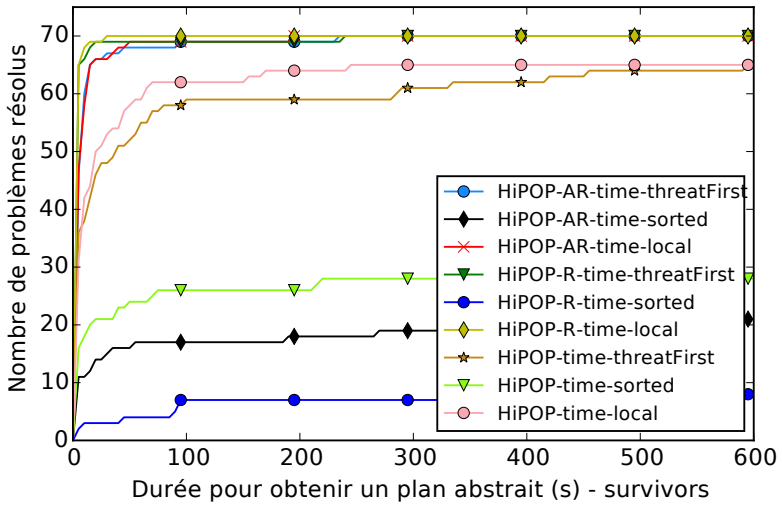


Figure 3.14 – Nombre de problèmes pour lequel un plan abstrait a été trouvé sur le domaine *survivors*. Comparaison des utilisations de h^{add}, h^{add}_{reuse}, h^{add}_{areuse}, *sorted* et *local*. Ce graphique représente en ordonnée le nombre de problèmes pour lesquel un plan abstrait a été trouvé en utilisant au plus le temps indiqué en abscisse.

lors de l'instanciation de chaque action abstraite, l'état complet du monde est connu car toutes les actions précédentes sont instanciées. Cela revient à utiliser l'heuristique *earliest* pour les défauts abstraits, dont les résultats sont présentés ci dessous.

Dans tous les cas, on observe une forte variabilité des résultats en fonction des spécificités de chaque domaine. Le nombre de problèmes résolus peut varier du simple au triple par l'utilisation de la bonne heuristique et aucune heuristique ne permet de résoudre tous les problèmes. Dans la suite, nous avons choisi de comparer les résultats à 3 configurations présentées ici, choisies pour avoir la meilleure configuration sur chaque domaine des IPC :

– HiPOP-AR-time-local

– HiPOP-R-time-local

– HiPOP-time-threatFirst

Quelle est l'influence de l'heuristique *earliest* ? La figure 3.15 présente les résultats de différentes configurations de HiPOP sur les 5 domaines présentés. On a ajouté aux configurations précédemment sélectionnées les meilleures configurations utilisant *earliest* et une configuration utilisant *sorted* sans *earliest*. On peut faire plusieurs observations.

Sur les domaines des IPC, il y a peu de différence entre une configuration utilisant *local* et une configuration utilisant *local* et *earliest*. Seule la configuration utilisant *areuse* est présente sur les figures pour ne pas les surcharger. En effet, si les liens ouverts sont déjà classés en fonction de leur ordre d'arrivée dans le plan, les classer en plus par date dans le plan ne change pas l'ordre de résolution car les actions des domaines des IPC n'incluent pas d'actions ayant des préconditions nécessaires à la fin de l'action (elles sont toutes nécessaire au début de l'action). Les actions hiérarchiques ne nécessitant pas de planifier avec plusieurs niveaux de raisonnement, l'ordre de résolution de ces actions ne

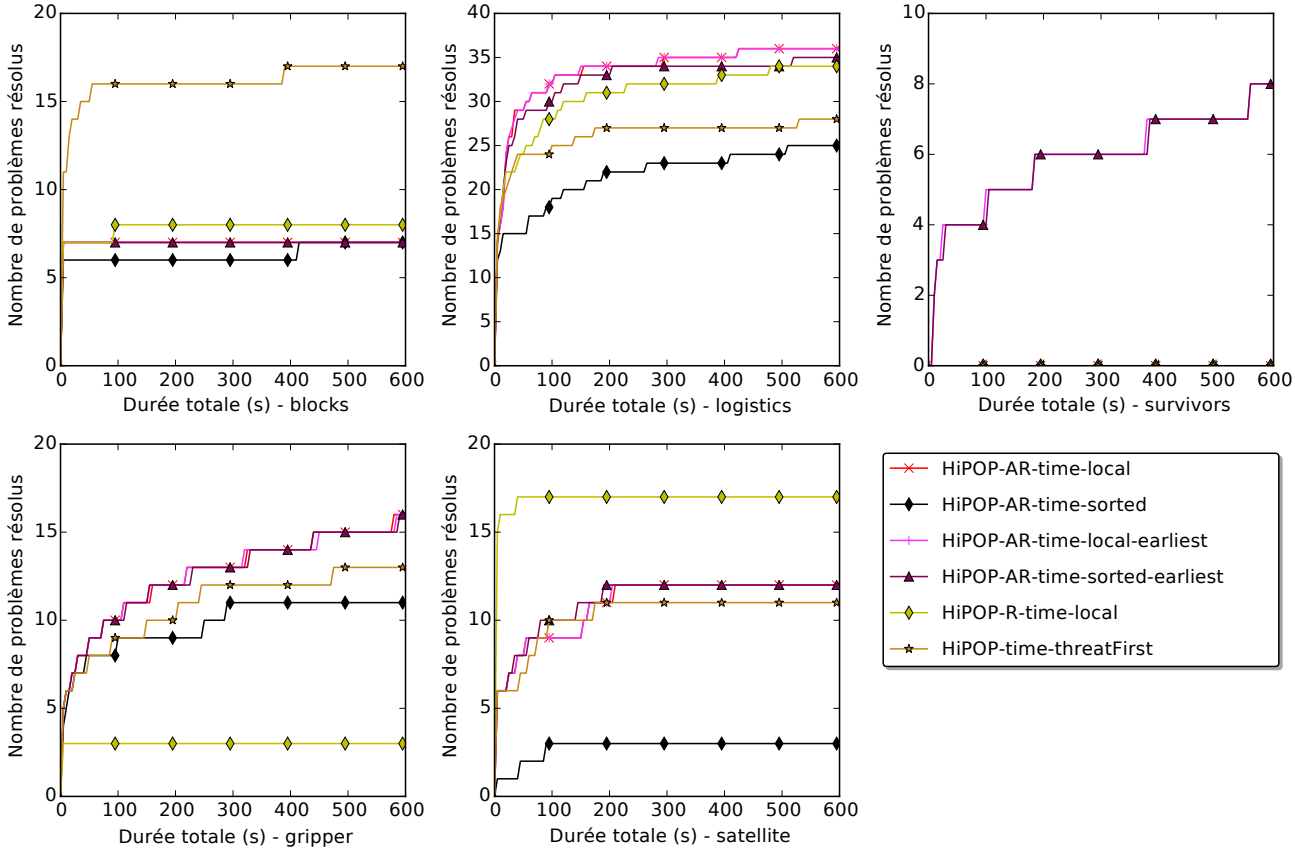


Figure 3.15 – Nombre de problèmes résolus par l'utilisation de l'heuristique *earliest* sur différents domaines. Comparaison de l'utilisation des variantes utilisant ou non *earliest*. Chaque graphique correspond à un domaine différent et représente en ordonnée le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse.

change visiblement pas significativement les résultats. A l'inverse, lors de l'utilisation de *sorted*, tous les liens ouverts sont considérés en même temps. Si on utilise en même temps *earliest*, et en cas d'égalité des coûts, on privilégie les liens ouverts les plus tôt dans le plan. Cela revient souvent à régler ceux introduits récemment en priorité : en effet si on résout un lien ouvert en introduisant une action, cette nouvelle action ajoute des liens ouverts et doit se situer avant le lien ouvert résolu. Ces nouveaux liens ouverts sont donc considérés en priorité à la fois par l'heuristique *local* et *earliest*. Dans les résultats, on observe effectivement que l'heuristique *sorted-earliest* est proche de l'heuristique *local* sur les domaines *satellite* et *logistics*.

La deuxième observation concerne le domaine *survivors*. La résolution des actions abstraites dans l'ordre chronologique (avec *earliest*) permet à HiPOP de résoudre certains problèmes alors qu'il en était incapable sans. Les deux meilleures configurations sont représentées sur la figure 3.15 et résolvent 8 problèmes au maximum. Les plans abstraits sont toujours trouvés rapidement et maintenant les solutions sont accessibles à la recherche, mais elle ne parvient pas à les transformer en plans valides. Une analyse plus poussée montre que la recherche se perd dans l'exploration de « plateaux » à cause d'une mauvaise évaluation du coût d'un plan. Cette mauvaise évaluation ainsi que la multitude de déplacements

possibles pour un robot mènent à une explosion combinatoire de l'espace à parcourir.

Exemple : Explosion combinatoire de la recherche

Reprenons l'exemple présenté page 81, et plus précisément le plan présenté à la figure 3.8d et rappelé ci-dessous.

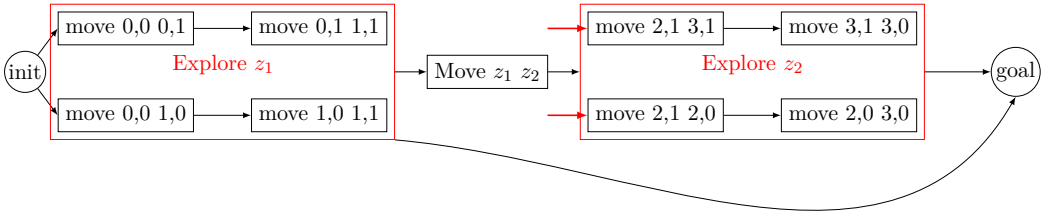


Figure 3.16 – Plan en cours d'instanciation présentant une difficulté à HiPOP.

En partant de ce plan, un des deux liens ouverts va être sélectionné. Mais l'évaluation qui en est faite par h^{add} se base sur l'état initial. Si la position du lien ouvert est plus proche de la position initiale que de la dernière position connue, alors l'ajout d'une action de déplacement fera diminuer l'évaluation du plan et la recherche aboutira. Par contre si cette position souhaitée est proche de l'état initial, son coût estimé sera faible. Si l'action de déplacement nécessaire est longue, l'estimation du plan augmentera. HiPOP chechera donc un plan correspondant plus à son évaluation initiale : il va étudier toutes les possibilités de déplacement allant vers la position souhaitée jusqu'à se rendre compte qu'il n'existe pas plus court que de se rendre directement de la dernière position connue à la position souhaitée.
Si ce processus se répète avec le deuxième robot, HiPOP aura gardé en mémoire tous les chemins alternatifs pour le premier et explorera donc en parallèle toutes les alternatives possibles de chemin pour les deux robots. L'explosion de la recherche est donc combinatoire. C'est ce qui empêche HiPOP de transformer les plans abstraits trouvés en plans valides.
Cet effet peut être encore renforcé si plusieurs équipes travaillent en parallèle dans le plan. Lors de la planification du déplacement de l'équipe la moins chargée de travail, la durée du plan n'évolue pas quand de nouvelles actions sont ajoutées (tant que la durée des tâches de l'autre équipe est suffisante). Cela contribue aussi à créer un « plateau » d'exploration : l'évaluation du plan n'évolue pas alors qu'on ajoute de nouvelles actions. Le planificateur doit alors considérer tous les plans ayant ce coût avant de pouvoir passer à autre chose.

Est-ce que l'option *time* permet d'optimiser efficacement la durée des plans trouvés ? Les figures 3.17 et 3.18 présentent les résultats des meilleurs configurations utilisant l'heuristique *time* (comme présenté à la page 91) avec les meilleures configurations d'HiPOP ne les utilisant pas. La figure 3.17 présente le nombre de problèmes résolus en un temps donné. La figure 3.18 présente la taille des plans trouvés par les meilleurs

configurations non temporelles en fonction de la configuration temporelle trouvant le plus de plans sur deux domaines spécifiques.

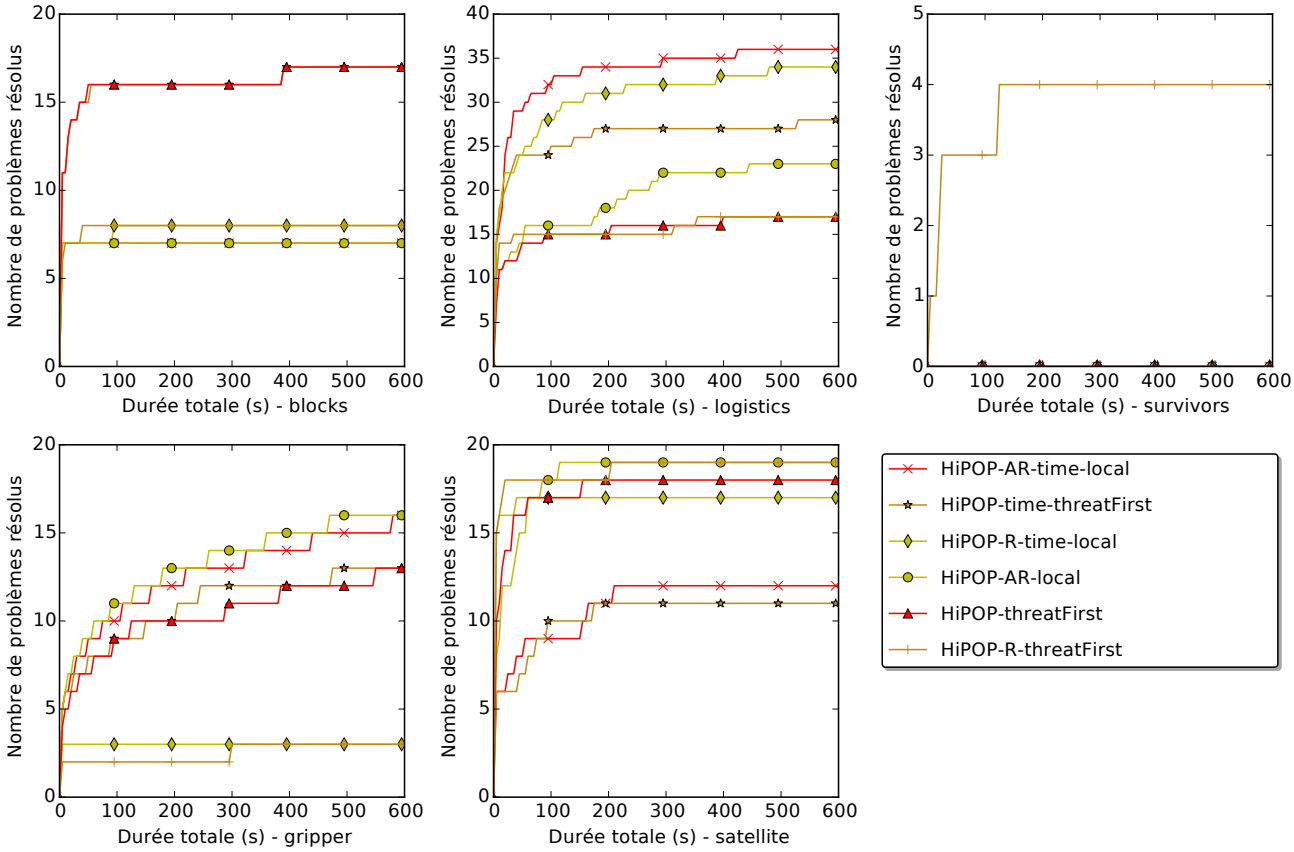


Figure 3.17 – Comparaison de l'utilisation ou non de l'heuristique *time* sur le nombre de problèmes résolus par HiPOP. Chaque graphique correspond à un domaine différent et représente en ordonnée le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse.

Premièrement on peut remarquer que cette heuristique peut aussi changer le nombre de problèmes résolus : elle n'a pas qu'un effet limité à la taille du plan produit. En changeant l'évaluation des plans, elle change l'ordre de résolution des plans et entraîne une recherche différente. Par exemple l'ajout d'une action pour résoudre un lien ouvert, si elle peut être exécutée en parallèle d'une action existante, ne va pas modifier la durée du plan alors qu'elle modifie le nombre d'actions du plan.

Sur les domaines *blocks* et *gripper*, les actions sont toutes de durée 1 et doivent se faire majoritairement séquentiellement. Le domaine n'a pas été prévu pour le parallélisme et l'heuristique *time* ne change pas fondamentalement le comportement de l'algorithme. Par contre sur les domaines *logistics* et *satellite*, la différence est visible. Dans un cas, l'heuristique *time* permet de résoudre plus de problèmes (*logistics*) alors que dans l'autre elle le réduit (*satellite*). Sur le domaine *survivors*, une configuration permet de résoudre certains problèmes. La métrique utilisée (ici le nombre d'actions) augmente à chaque nouvelle introduction d'actions contrairement à l'heuristique *time*. Cela permet de limiter le nombre de plans ayant la même évaluation heuristique et permet donc au planificateur de fouiller plus facilement tous les plans ayant un coût donné pour pouvoir passer aux

plans ayant un coût supérieur et menant à une solution. Mais même dans ce cas, on ne trouve pas beaucoup de solutions sur ce domaine.

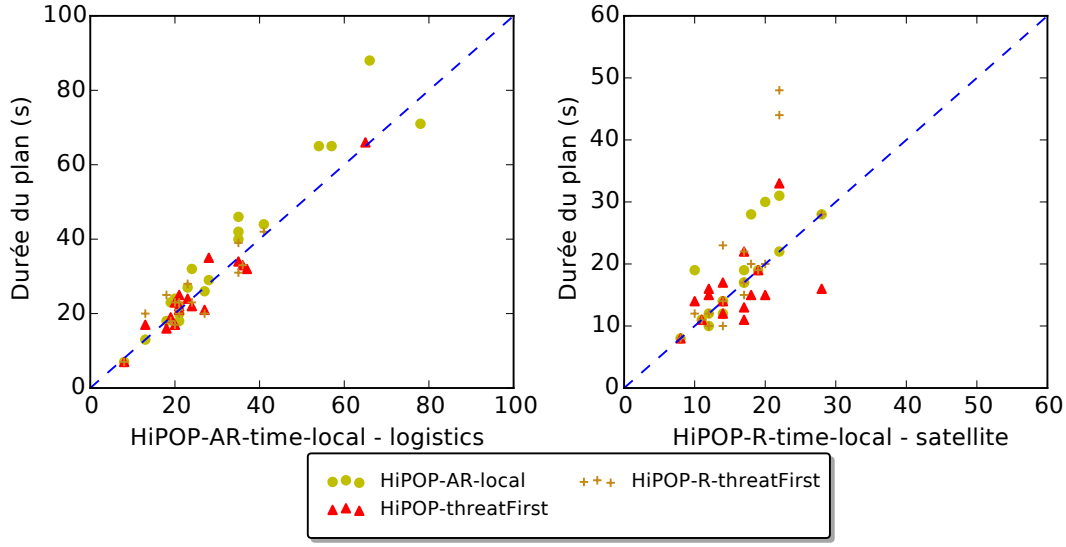


Figure 3.18 – Comparaison de l'utilisation ou non de l'heuristique *time* sur la qualité des plans trouvés par HiPOP. Chaque graphique correspond à un domaine différent et représente en ordonnée la taille des plans trouvées par une des heuristique non temporelle en fonction de la taille du plan trouvé par l'heuristique temporelle ayant résolu le plus de problème sur ce domaine. Un point au-dessus de la première bissectrice signifie donc un plan plus court produit par la configuration utilisant *time*.

La figure 3.18 présente la durée des plans trouvés. Sur les domaines *blocks* et *gripper*, les plans étant majoritairement séquentiels, il n'y a pas de différence entre l'utilisation ou non de l'heuristique *time* car dans les deux cas on cherche à optimiser un critère quasiment identique. Sur le domaine *survivors* il n'existait pas de référence temporelle. On a donc représenté uniquement les résultats pour *logistics* et *satellite*.

Dans ces deux domaines, certains problèmes sont résolus avec des plans plus courts et d'autres avec des plans plus longs. Sur *logistics* une majorité de plans sont plus courts quand ils sont calculés avec l'heuristique *time* (qui de plus en résout plus). Sur *satellite* la différence est moindre. Le nombre de points au-dessus de la bissectrice est néanmoins supérieur au nombre de points en dessous de la bissectrice.

On voit donc que l'heuristique *time* peut avoir plusieurs effets différents en fonction du domaine. Elle permet en moyenne de diminuer la taille des plans calculés, mais sur le domaine qui nous intéresse le plus (celui qui concerne une mission multirobot), les résultats ne sont pas concluants car il n'y a pas suffisamment de problèmes résolus.

Quel est l'apport de l'utilisation des actions hiérarchiques pour la planification ? La figure 3.19 présente les résultats des meilleures configurations utilisant les actions hiérarchiques (comme présenté à la page 91) avec les meilleures configurations d'HiPOP ne les utilisant pas. On a aussi inclus deux configurations n'utilisant pas l'heuristique *time* pour faire la comparaison la plus générale possible, en particulier sur *survivors*.

L'utilisation des actions hiérarchiques permet de résoudre beaucoup plus de problèmes

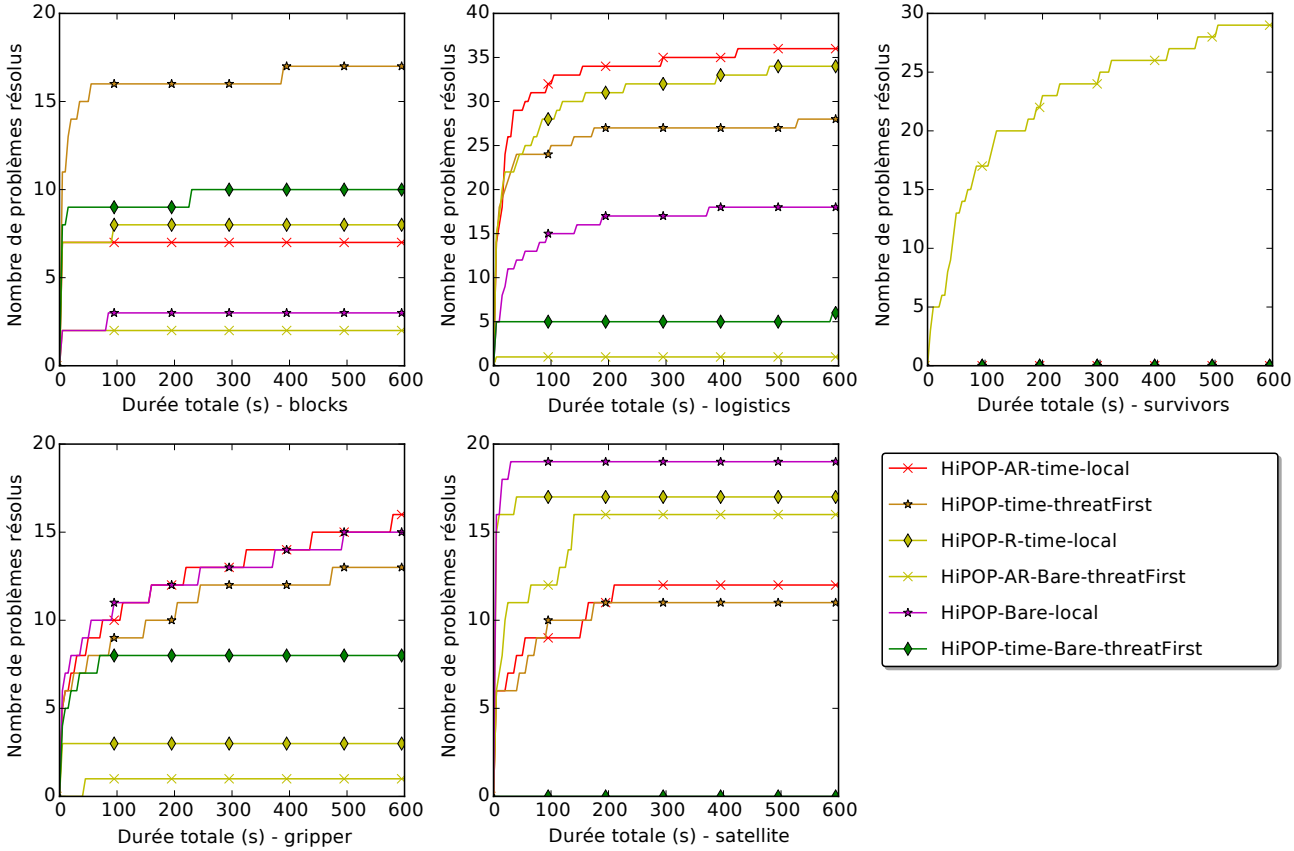


Figure 3.19 – Comparaison de l'utilisation ou non des actions hiérarchiques sur le nombre de problèmes résolus par HiPOP. Les configurations auxquelles aucune action hiérarchique n'a été fournie sont indiquées par *Bare*. Chaque graphique correspond à un domaine différent et représente en ordonnée le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse.

sur les domaines *blocks* et *logistics*. Sur *gripper*, les deux configurations sont équivalentes et sur *satellite* les meilleures configurations n'utilisent pas d'action hiérarchique. Sur *survivors*, les configurations utilisant l'heuristique *time* ne trouvent aucune solution. Elles sont donc moins performantes que l'alternative d'utiliser des actions hiérarchiques avec l'heuristique *earliest*.

On voit donc un apport très net des actions hiérarchiques dans les domaines qui s'y prêtent : elles permettent le raisonnement sur plusieurs véhicules dans *logistics* pour transporter un unique paquet par exemple. Au contraire, dans *gripper* et *satellite* les actions hiérarchiques ne font que regrouper plusieurs actions qui apparaissent ensemble dans le plan parce qu'elles y sont forcées par le domaine. Par exemple dans le domaine *satellite* l'action qui consiste à pointer un instrument vers un point de calibration, l'allumer puis le calibrer ne fournit pas d'information supplémentaire au planificateur. Pour calibrer un instrument, il faut nécessairement l'allumer avant et l'avoir pointé dans une direction de calibration. Dans ces cas-là, les actions hiérarchiques au mieux aident à la marge le planificateur en lui faisant réaliser plusieurs opérations en même temps et en limitant le nombre de branches à explorer, mais dans certains cas elles peuvent le ralentir ou l'empêcher de trouver certaines solutions.

Dans le cas de *survivors*, lorsque l'on ne cherche pas à optimiser la durée de la mission, l'ajout de nouvelles actions de déplacement fait augmenter le coût du plan. On limite donc la création de « plateaux », ce qui permet d'éviter une partie des problèmes rencontrés précédemment. Mais cette explication n'est pas suffisante pour expliquer la résolution de presque 30 problèmes par une configuration n'utilisant pas les actions hiérarchiques alors qu'on a vu précédemment qu'avec des actions hiérarchiques et sans l'utilisation de *time* on résolvait moins de 5 problèmes Cela provient du fait que l'utilisation des actions hiérarchiques contraint les robots à des déplacements particuliers qui peuvent ne pas être bien gérés par les heuristiques. Dans le cas sans *time*, on a un unique robot qui fait toutes les explorations par exemple, cette solution n'est considérée comme bonne que dans le cas où on cherche à minimiser le nombre d'actions dans le plan. Si on utilise *time*, on cherche à optimiser la durée du plan et ce plan est donc de mauvaise qualité. Lors de l'utilisation des actions hiérarchiques, on ne peut pas explorer toutes les cellules de la plus éloignée à la plus proche de l'état initial : les actions hiérarchiques imposent une autre forme à la solution. Cette forme est sujette aux problèmes présentés dans l'exemple page 93

On voit donc que dans les domaines où il existe une hiérarchie possible des actions, l'apport est important. Mais elle ne fonctionne pas avec tous les domaines et dépend de la structure de celui-ci et de la modélisation des actions hiérarchiques. En particulier, elle semble limiter le nombre de plans trouvés dans *survivors*.

Comment HiPOP se compare-t-il avec d'autres planificateurs ? Les figures 3.20 et 3.21 présentent les résultats des meilleures configurations d'HiPOP (comme présenté à la page 91) avec d'autres planificateurs temporels. La figure 3.20 présente le nombre de problèmes résolus en un temps donné. La figure 3.21 présente la taille du plan produit pour chaque problème. Une courbe au-dessus d'une autre signale donc un planificateur qui a trouvé un plan plus long pour un problème donné. Au vu des différences observées, l'axe des abscisses est gradué en échelle logarithmique.

Coté nombre de problèmes résolus, YAHSP tout d'abord se démarque clairement des autres planificateurs. Il réussit à trouver une solution très rapidement à chaque problème et cela dans tous les domaines. YAHSP a été le vainqueur des IPC 2014 dans la catégorie Agile : le but était de trouver une solution le plus vite possible, sans s'intéresser à la qualité du plan fourni. Si on s'intéresse à la durée du plan trouvé, on voit que cette vitesse de résolution a un coût en terme de qualité de la solution. Les solutions de YAHSP sont bien plus longues (la figure 3.21 est en échelle logarithmique), souvent au moins 5 fois plus longues. Dans le domaine *gripper* par exemple, les solutions trouvées proposent de prendre et de déposer immédiatement dans la même salle toutes les balles disponibles à chaque retour à la salle de départ et pour finalement n'en garder que deux. La figure inclut aussi YAHSP en mode anytime : tant que les dix minutes ne sont pas écoulées, il continue la recherche pour essayer d'améliorer le plan. Même si on note une amélioration du plan, on reste loin des plans trouvés par les autres planificateurs (quand ils arrivent à résoudre les problèmes).

TFD de son coté arrive à résoudre de nombreux problèmes dans les domaines *blocks* et *satellite* mais très peu dans les deux autres domaines. Les plans qu'il trouve sont de bonne qualité, en général meilleurs que ceux trouvés par HiPOP. HiPOP par contre ne recherche pas dans le même espace : il cherche uniquement parmi les plans qui respectent la hiérarchie des actions qui lui est imposée. Certains plans plus courts peuvent donc être trouvés par TFD mais pas par HiPOP.

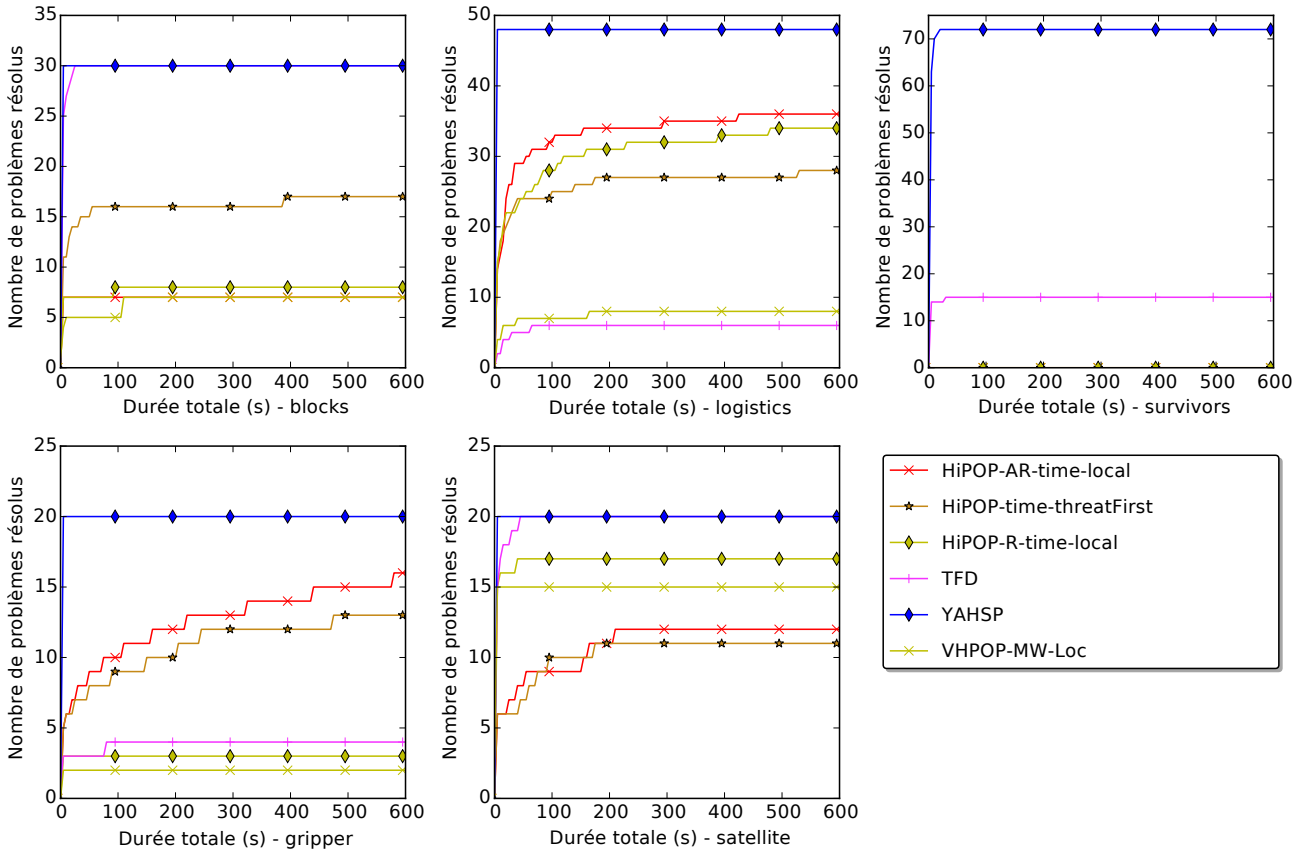


Figure 3.20 – Comparaison de plusieurs planificateurs sur le nombre de problèmes résolus. Comparaison des meilleures configurations d'HiPOP avec les autres planificateurs en terme de nombre de solutions trouvées.

Dans les tests que nous avons faits, VHPOP a été configuré pour utiliser uniquement une heuristique proche de *local* : tri des liens ouverts par coût et priorité aux actions venant de la dernière action ajoutée. La version ayant concouru aux IPC utilise quatre heuristiques en parallèle mais nous avons cherché à avoir une référence utilisant un algorithme de recherche POP classique avec une heuristique proche d'une des nôtre. Dans cet ensemble de domaines, VHPOP résout moins d'instances que les autres planificateurs dans le temps imparti. Sur le domaine *logistics*, comme avec TFD, il trouve des solutions de meilleure qualité qu'HiPOP mais en prenant plus de temps.

Dans l'ensemble, HiPOP se situe dans un compromis entre la vitesse de recherche de YAHSP et la qualité des solutions produites par les autres planificateurs. Cela provient de l'utilisation des actions hiérarchiques qui permettent de limiter le domaine de recherche. Les solutions les plus courtes peuvent ne plus être accessibles (si elles ne se conforment pas à ces actions hiérarchiques) mais l'espace de recherche est aussi plus restreint ce qui accélère la recherche. Par contre, cela nécessite un travail de modélisation supplémentaire de la part de l'opérateur et un domaine qui se prête à une telle modélisation.

Conclusion Les résultats précédents montrent l'influence du domaine sur les performances des algorithmes. Les propriétés particulières de chaque domaine influencent l'efficacité des différentes heuristiques. En particulier, bien que HiPOP permette de résoudre un

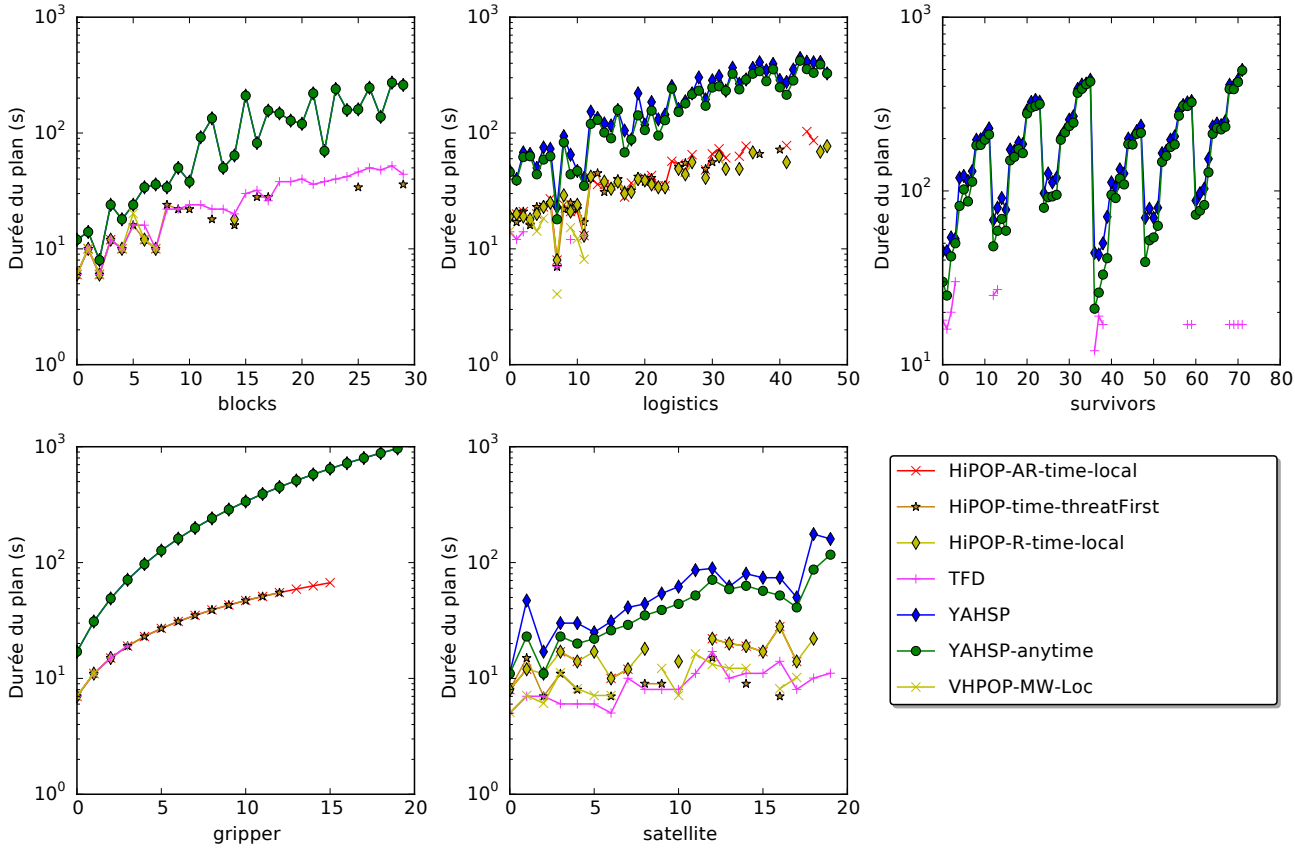


Figure 3.21 – Qualité des plans trouvés par plusieurs planificateurs. Comparaison des meilleures configurations d'HiPOP avec les autres planificateurs en terme de durée du plan trouvé. Au vu des différences constatées, les courbes sont tracées en échelle logarithmique.

certain nombre de problèmes sur des domaines multirobots comme *logistics*, il se trouve quasiment incapable d'utiliser les actions hiérarchiques fournies pour résoudre les problèmes du domaine *survivors*. Une analyse rapide a montré que le blocage intervient avec l'obtention du premier plan abstrait : HiPOP a du mal à passer d'un plan abstrait à un plan valide. S'il savait le faire, alors il pourrait résoudre tous les problèmes de *survivors* rapidement (car certaines configurations trouvent un plan abstrait pour tous les problèmes en moins de une minute).

Conclusion

Ce chapitre présente l'algorithme de planification hybride nommé HiPOP.

La planification hybride a été choisie dans l'optique de trouver des plans plus « riches » pour être plus facilement exécutables et réparables. Un implémentation a été faite, utilisant une description des actions hiérarchiques dans un langage proche de PDDL. Plusieurs heuristiques utilisées lors de la sélection des plans ou des défauts ont été reprises de la littérature et adaptées à l'utilisation d'actions hiérarchiques.

Les résultats ont montré que HiPOP était capable de trouver des solutions sur une variété de problèmes. L'utilisation des actions hiérarchiques n'est pas nécessaire mais

permet d'améliorer grandement les performances de la planification dans les domaines permettant une hiérarchisation des actions. Les domaines issus des IPC sont basés sur des problèmes d'algorithmique classiques et il est souvent difficile de les décrire avec des actions hiérarchiques. Le domaine *logistics* par exemple a du être adapté pour présenter une hiérarchie des actions plus claire et les gains ont été plus importants que sur les autres domaines.

La plus grande surprise vient néanmoins des résultats obtenus sur le domaine *survivors*. Bien qu'un plan abstrait soit trouvé rapidement, la transformation en plan valide n'est que très rarement réussie. Plus précisément, c'est le raisonnement sur les positions individuelles des robots, entre les différentes patrouilles, qui n'aboutit pas.

La prochaine étape est donc de nous intéresser au raisonnement géométrique dans un planificateur symbolique comme HiPOP.

4

Mise en place d'un raisonnement géométrique dans un planificateur symbolique

Sommaire

Le but de ce chapitre est d'améliorer les performances (en temps de calcul, nombre de plans trouvés et qualité de la solution) en utilisant certaines propriétés géométriques des problèmes étudiés.

La première section va analyser un exemple simple pour mettre en évidence le raisonnement utilisé et les différents facteurs d'amélioration possibles. Les sections suivantes présenteront les différentes améliorations qui ont été proposées pour tirer parti des spécificités du raisonnement géométrique. La dernière section présentera les résultats obtenus sur les benchmarks déjà utilisés au chapitre 3.

4.1 Analyse des performances de HiPOP et mise en évidence de l'importance du raisonnement géométrique

4.1.1 Analyse poussée sur un exemple

Exemple : Résolution d'un problème géométrique avec un planificateur symbolique

On cherche à résoudre le problème présenté à la figure 4.1.

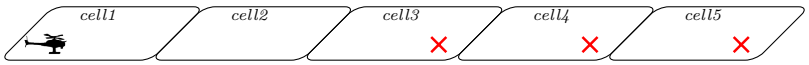


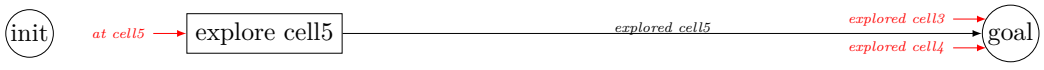
Figure 4.1 – Exemple d'un problème illustrant l'intérêt du raisonnement géométrique. Un robot aérien doit explorer 3 cellules alignées. Le mouvement entre chaque couple de cellule est possible (pas uniquement entre cellules adjacentes). La cellule initiale est à une distance 2 de la première cellule à explorer.

La durée d'un déplacement entre 2 cases adjacentes est de 1. La durée d'une exploration est aussi de 1. Le plan optimal fait donc 7 de long si on explore les cellules dans l'ordre `cell3`, `cell4`, `cell5`. La description en PDDL de ce problème est disponible à l'annexe C. On le résout en utilisant les heuristiques *time*, *local* et *areuse*. HiPOP va trouver un plan de longueur 9. La figure 4.2 présente les principales étapes de la résolution.
La recherche commence avec un plan vide.

Figure 4.2 – Exemple de résolution d'un problème géométrique avec un planificateur symbolique.

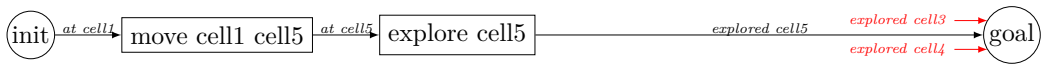
(a) Plan vide.

Le lien ouvert ayant le coût le plus élevé est `explored cell5`. On commence donc par le résoudre et il n'y a qu'une seule possibilité. On aboutit alors au plan suivant.



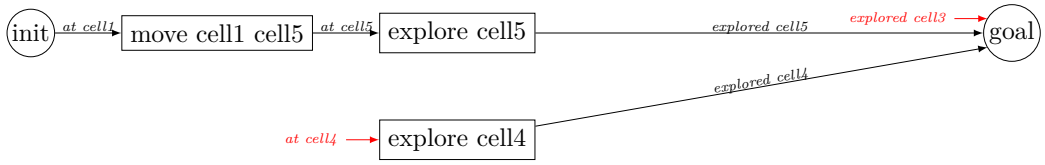
(b) Le premier but en cours de résolution : l'action d'exploration a été ajoutée.

L'heuristique *local* va privilégier la résolution du lien causal introduit le plus récemment. Pour le résoudre la meilleure solution est d'introduire une action de déplacement depuis l'action initiale. Le lien ouvert de cette nouvelle action est résolu en la liant à l'état initial. On aboutit alors au plan suivant.



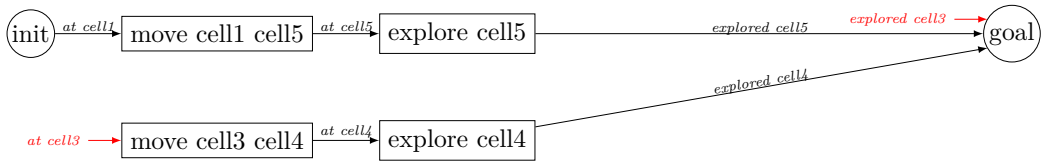
(c) Le premier but est résolu.

Le prochain lien ouvert a être choisi est `explored cell4`. Une seule solution pour le résoudre est possible : l'ajout de l'action d'exploration. On aboutit alors au plan suivant :



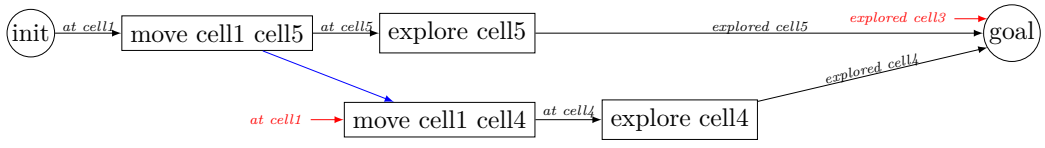
(d) Le premier but est résolu et l'action d'observation pour le deuxième est introduite. Il n'y a aucune contrainte temporelle sur `explore cell4`.

Le prochain lien ouvert à être exploré est `at cell4`. Introduire un déplacement depuis `cell1` est directement détecté comme une menace : il efface `at cell1` et donc nécessite d'être après `move cell1 cell5`. Par contre introduire un déplacement depuis `cell3` n'est pas détecté comme une menace.



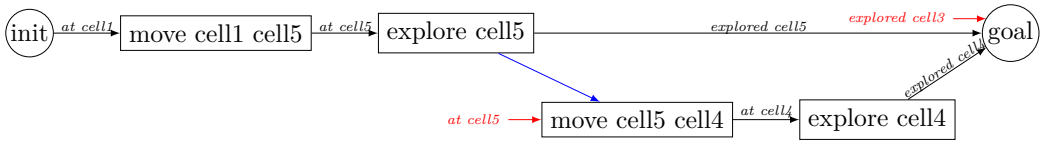
(e) Aucune menace n'est détectée dans ce cas de figure.

Il est nécessaire de chercher à résoudre le nouveau lien ouvert pour introduire des actions de déplacement qui vont soit trop allonger le plan (ce qui rendra l'évaluation heuristique trop grande et cette branche ne sera plus explorée) soit être détectées comme des menaces. Dans tous les cas, il finira par déduire que le mouvement vers `cell4` ne peut avoir lieu qu'après `cell5` à cause des liens causaux déjà existants. Mais cela nécessitera l'exploration d'un certain nombre de plans. Ce nombre de plans étant proportionnel au nombre de positions disponibles, cette exploration peut devenir rédhibitoire dans les problèmes de grande taille.
Une fois cette déduction faite, on se retrouve à explorer les plans suivants.



(f) La flèche bleue représente une contrainte de précédence temporelle.

Dans ce cas de figure l'action de déplacement de `cell1` à `cell4` a été correctement placée après le déplacement vers `cell5`. Néanmoins, l'estimation du coût du lien ouvert `at cell3` est faite avec h^{add} et ne dépend donc que de l'état initial. Comme c'est la position initiale du robot, son coût est estimé à 0. Or dans ce cas le robot ayant déjà fait le déplacement vers `cell5`, il faudrait au moins 4 unités de temps pour y aller : ce plan est bien plus mauvais qu'estimé actuellement par l'heuristique. Le plan qui sera finalement produit part du plan partiel suivant.



(g) La flèche bleue représente une contrainte de précédence temporelle.

Le dernier lien ouvert introduit peut être résolu en se liant directement à `move cell1 cell5`. Le planificateur va procéder de la même façon pour le dernier objectif et le plan produit va donc consister à commencer par explorer `cell5` avant de retourner en arrière pour explorer `cell4` puis `cell3`. Le plan produit est donc sous-optimal : sa durée est de 9 alors que le plan optimal a une durée de 7. Ces deux plans sont représentées sur la figure 4.3.

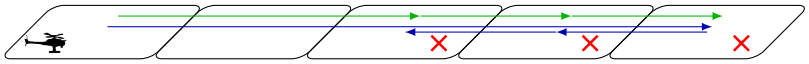


Figure 4.3 – Plans solutions du problème illustrant l'intérêt du raisonnement géométrique. Le plan représenté en vert montre le plan optimal : les cellules sont observées dans l'ordre optimal. Le plan représenté en bleu montre le plan calculé par HiPOP : les cellules sont observées dans un ordre sous-optimal.

On voit sur cet exemple que le raisonnement uniquement symbolique ne convient pas à la résolution d'un problème géométrique, même simple comme celui-là.

HiPOP en configuration *local-areuse-earliest-time* résout le problème donné en exemple en environ 100ms en explorant 80 plans et trouve un plan de longueur 9 (cf. tableau 4.1). La longueur indiquée de 9.01 correspond à la durée des 9 actions et des 10 intervalles nécessaires pour séparer temporellement ces actions entre elles.

Moyenne sur 20 essais	Plans explorés	Durée de la recherche (ms)	Longueur du plan
No motion	80.00	93.30	9.01

Tableau 4.1 – Résultats comparatifs sur un exemple en utilisant la configuration HiPOP-*local-areuse-earliest-time*. L'intitulé « no motion » indique que les positions ne sont pas traités différemment des autres fluents.

Nous allons donc nous intéresser aux différentes caractéristiques des fluents représentant la position des robots afin de pouvoir en tirer parti par la suite.

4.1.2 Caractérisation et détection des fluents de position

Dans une représentation symbolique, les positions sont représentées par un ensemble de fluents. Chaque fluent représente le fait qu'un robot est à une position donnée. Dans l'exemple donné précédemment, ces fluents sont `at aav1 cell1`, `at aav1 cell3`, etc. Pour le planificateur, il n'y a pas plus de liens entre `at aav1 cell1` et `at aav1 cell3` qu'entre `at aav1 cell1` et `explored cell5`. Sur la figure 4.2e par exemple, le planificateur ne peut pas détecter que `at aav1 cell1` et `at aav1 cell3` sont incompatibles entre eux. Il doit essayer d'ajouter les actions de déplacement nécessaires pour établir `at aav1 cell3` pour s'en rendre compte.

Une des propriétés fondamentales est qu'un robot ne peut être qu'à une position à un moment donné. Tous les fluents représentant la position d'un robot sont donc liés par une contrainte : un seul au maximum peut être vrai à un instant donné. C'est cette propriété qui permettra de détecter que `at aav1 cell1` et `at aav1 cell3` sont incompatibles entre eux.

Un autre aspect important est la présence des actions de déplacement. Dans tous les domaines étudiés présentant des fluents de position (*satellite*, *survivors*, *logistics*), des actions de déplacement respectant le même format étaient disponibles. Ces actions ont pour unique précondition la position d'un robot et pour effet de changer la position en une autre. Cela signifie qu'il est toujours possible d'établir les positions souhaitées des robots sans avoir à raisonner sur le reste des fluents : quel que soit l'état du monde, on peut déplacer un robot vers n'importe quelle position souhaitée. De plus, ces actions sont les moyens les plus courts d'atteindre une position : cette action représente le déplacement le plus direct possible entre deux points.

La définition 42 définit les fluents de position comme étant les fluents vérifiant ces deux propriétés. Pour garantir la première propriété (l'unicité d'un fluent à tout moment) il faut que cette propriété soit vraie à l'état initial (point 1) et que l'application de toute action conserve cette propriété. Toute action qui efface la position actuelle doit alors en créer une nouvelle (point 2). Toute action qui crée une nouvelle position doit alors effacer la précédente en s'assurant que c'est bien la position actuelle (point 3). Les fluents de position doivent aussi être associés à des actions de déplacement (c'est la deuxième propriété). Pour chaque couple de positions, une action passant d'une position à l'autre doit exister (point 4). Cette action doit être le plus court moyen de passer d'une position à une autre (points 5 et 6). On définit alors formellement :

Définition 42 : Fluents de position

Soit $\langle F, A, I, G\rangle$ un problème de planification hybride.
L'ensemble de fluents $\mathcal{P} \subset F$ est une famille de fluents de position ssi. :

1. $I \cap \mathcal{P}$ a un seul élément.
2. $\forall a \in A, \forall f \in \mathcal{P},\ f \in Del(a) \implies f \in Pre(a) \wedge \exists g \in \mathcal{P} \mid g \neq f \wedge g \in Add(a)$.
3. $\forall a \in A, \forall f \in \mathcal{P},\ f \in Add(a) \implies \exists g \in \mathcal{P} \mid g \neq f \wedge g \in Pre(a) \wedge g \in Del(a)$.
4. $\forall from, to \in \mathcal{P}^2 \mid from \neq to, \exists a \in A \mid Pre(a) = \{from\} \wedge Add(a) = \{to\} \wedge Del(a) = \{from\}$. On note cette action $a^{from \to to}$.
5. $\forall a \in A, \exists from, to \in \mathcal{P}^2 \mid from \in Pre(a) \wedge to \in Add(a) \implies dur(a) \leq dur(a^{from \to to})$.

6. Les durées des actions $a^{from \to to}$ doivent respecter l'inégalité triangulaire : $\forall f, g, h \in \mathcal{P}^3,\ dur(a^{f \to g}) + dur(a^{g \to h}) \leq dur(a^{f \to h})$.

Ces propriétés peuvent être vérifiées automatiquement en une seule itération sur chaque action. La plus grande difficulté est d'avoir un bon candidat $\mathcal{P}$.

Avec un formalisme utilisant une description par variable d'état (voir 1.1.6), les ensembles candidats peuvent être directement extraits de la description du monde. En effet dans ce formalisme une variable est définie avec son *domaine* (*i.e.* l'ensemble des valeurs qu'elle peut prendre) donc chaque domaine est un ensemble candidat pour définir une famille de fluents. De plus, par construction la première propriété est vérifiée : on est sur qu'il n'y a qu'une seule seule valeur vraie de cette variable à tout instant.

Dans une description par fluents comme celle que nous utilisons, nous avons dû faire des hypothèses pour tester uniquement un petit nombre d'ensembles candidats. Nous avons donc choisi de ne tester que les fluents provenant de prédicats à deux variables et de les grouper selon l'instanciation de leur première variable. Ainsi le prédicat `at ?r ?l` va définir une famille potentielle de fluents de position pour chaque instanciation possible du robot `?r`. Cette hypothèse permet de correctement détecter les fluents de position dans les domaines présentés jusque-là car les prédicats y sont tous décrits selon le même modèle, mais cette hypothèse nécessiterait des adaptations pour traiter des problèmes plus généraux.

De plus la définition précédente permet d'avoir des états où la position d'un robot est indéfinie. C'est le cas dans la plupart des domaines : pendant un déplacement par exemple, la position du robot est indéfinie ce qui empêche de commencer une action qui a besoin d'une position précise du robot si une action de déplacement est en cours. La position n'est assurée qu'au début et à la fin de l'action. L'existence et l'unicité de la position de chaque robot sont donc assurées avant et après les actions, mais pas pendant certaines actions.

Les conditions posées sur la définition des actions de déplacement, en particulier le point 5 de la définition 42, impose qu'une action existe pour passer de n'importe quelle position à n'importe quelle autre position. Cela n'est pas toujours vérifié en pratique, où les domaines peuvent être non connexes (comme les exemples présentés au chapitre 2). Dans ce cas, il est nécessaire de pré-calculer les chemins les plus courts pour aller d'une position à une autre. Cette connaissance peut ensuite être fournie au planificateur sous forme d'actions élémentaires (modifiant le domaine) ou d'actions hiérarchiques.

Dans la suite de ce chapitre, nous allons voir comment utiliser les fluents de position. L'unicité des positions va permettre de détecter plus tôt des incompatibilités entre fluents. La présence des actions de déplacement va permettre de prendre en compte leur utilisation lors du calcul de h^{add}, lors du calcul des contraintes temporelles du plan et dans l'ordre de résolution des défauts.

4.2 Utilisation de l'unicité des positions

La première propriété qui nous intéresse est l'unicité des positions : il ne peut y avoir 2 positions différentes simultanées pour un robot. On parle alors d'exclusion mutuelle (ou *mutex*). Ces mutex vont permettre de détecter plus de menaces : on sait que la création d'un fluent ne peut pas avoir lieu au milieu d'un lien causal concernant un fluent avec lequel il est en mutex. Autrement dit : si on veut conserver une position, il n'est pas possible d'en créer une autre.

On se propose donc d'utiliser les mutex lors du calcul des menaces. Au lieu de détecter une menace uniquement quand une action efface un fluent établi par un lien causal, on va maintenant aussi détecter une menace quand une action crée un fluent qui est en mutex avec le lien causal.

Exemple : Utilisation des mutex pour la détection des menaces

En reprenant l'étape présentée à la figure 4.2e et rappelée ci-dessous, on peut mettre en évidence la différence que cette détection fait.

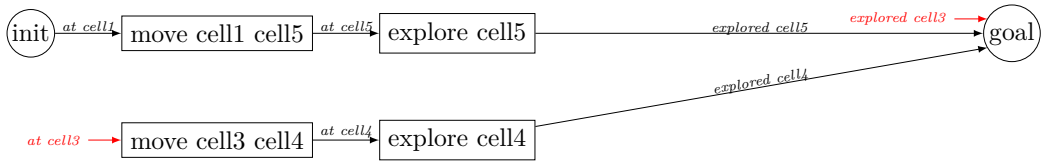


Figure 4.4 – Exemple de plan où les mutex sont utilisés pour détecter une menace en avance. Dans ce cas de figure, sans l'utilisation des fluents de position, aucune menace n'est détectée. En les utilisant, on détecte immédiatement que l'action `move cell3 cell4` ne peut être concurrente avec l'action de déplacement ou avec les liens causaux déjà dans le plan.

En utilisant les fluents de position (ici la famille des fluents de la forme `at *`), on détecte que l'action `move cell3 cell4`, qui crée le fluent `at cell4`, menace les liens causaux portant sur `at cell1` et `at cell5`. Comme cette action ne peut pas être placée avant l'état initial, elle doit être placée après `explore cell5`. Cela évite au planificateur de devoir essayer d'établir `at cell3` pour s'en rendre compte.

Cette utilisation des mutex est proche de l'utilisation des « conflits » tels que définis dans le chapitre 3 pour la description des actions hiérarchiques (cf. section 3.2.2). Les conflits permettent effectivement, sans avoir à détecter des familles de fluents de position, de calculer des menaces supplémentaires. Les exemples qui ont été donnés concernent d'ailleurs des fluents de position. Mais les conflits peuvent aussi concerner des fluents autres que des fluents de position, et les fluents de position ont d'autres utilisations que le calcul des mutex. Ces deux notions se rejoignent donc dans le cas d'utilisation des conflits sur des fluents de position mais sont complémentaires dans le cas général.

Une autre utilisation de la même propriété concerne la résolution des liens ouverts. En cas de lien ouvert, la méthode de résolution consiste à essayer d'établir un lien causal vers toute action passée produisant ce fluent (en produisant un plan différent par action) et d'essayer d'ajouter toutes les actions possibles qui produisent ce fluent en tant qu'effet principal (ici aussi un nouveau plan par action). Mais si le lien ouvert concerne un fluent de position, on sait qu'il est inutile d'utiliser les actions de déplacement trop anciennes : seule la dernière action précédant le lien causal peut être utilisée. Autrement dit, si une action a produit un fluent de position concerné par un lien ouvert, on ne considérera pas a pour la résolution du lien ouvert s'il existe une action produisant un fluent de position de la même famille entre a et le lien ouvert. Cela permet d'éviter d'avoir à explorer ces plans pour se rendre compte qu'ils sont forcément incohérents.

Exemple : Utilisation des mutex pour la résolution des liens ouverts

En reprenant l'étape présentée à la figure 4.2f et rappelée ci-dessous, on peut mettre en évidence l'utilité de cette utilisation.

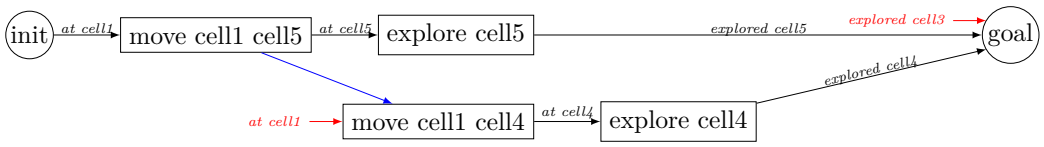


Figure 4.5 – Exemple de plan où les mutex sont utilisés lors de la résolution d'un lien ouvert. Dans ce cas de figure, sans l'utilisation des fluents de position, le lien ouvert vers `at cell1` peut être réglé en se liant avec l'action initiale.

En utilisant les fluents de position (ici la famille des fluents de la forme `at *`), on détecte que l'action `move cell1 cell5`, produit un fluent de position. Au moment de régler le lien ouvert vers `at cell1`, le planificateur ne va donc pas considérer la possibilité d'ajouter un lien causal de `init` à `move cell1 cell4`. Cela va éviter d'avoir à ensuite détecter une menace de `move cell1 cell5` sur ce nouveau lien causal et à se rendre compte qu'il n'existe pas de solution pour éliminer ce plan. Dans ce cas, on détecte en amont les plans invalides.

Le tableau 4.2 présente les résultats de la résolution de cet exemple avec les modifications proposées. Le nombre de menaces détectées augmente ce qui contribue à faire augmenter le nombre de plans explorés, mais la résolution des menaces (*i.e.* l'introduction d'une nouvelle contrainte temporelle) est plus rapide que la résolution d'un lien ouvert (qui nécessite l'ajout d'actions et possède plus de résolveurs). Le temps nécessaire à la recherche diminue donc.

Moyenne sur 20 essais	Plans explorés	Durée de la recherche (ms)	Longueur du plan
No motion	80.00	93.30	9.01
Mutex	86.00	85.75	9.01

Tableau 4.2 – Résultats comparatifs sur un exemple en utilisant la configuration HiPOP-*local-areuse-earliest-time*.

4.3 Utilisation de la présence des actions de déplacement

4.3.1 S'affranchir de l'utilisation exclusive de l'état initial dans h^{add}

Une autre utilisation potentielle de l'unicité des positions concerne l'évaluation du coût des fluents par h^{add}. En effet lors de la planification hybride, h^{add} est utilisé à partir de l'état initial pour estimer le coût d'établissement d'un fluent. Mais cette estimation ne prend pas en compte l'état actuel du plan. Or dans le cas de positions, une fois le premier déplacement effectué, cette estimation devient fausse. De plus, la présence des actions de

déplacement garantit qu'avec la connaissance de la dernière position d'un robot, on peut évaluer la durée minimale nécessaire pour atteindre une nouvelle position.

Nous avons donc modifié notre utilisation de h^{add}. Lors de l'évaluation d'un lien ouvert, si le fluent concerné est un fluent de position, alors le coût donné par h^{add} est ignoré et on utilise la durée de l'action de déplacement qui part du dernier fluent de position de cette famille crée et qui arrive à la position voulue. Si la dernière position n'est pas connue (parce qu'il y a plusieurs candidats possibles), on prend la durée minimum pour ne pas sur-contraindre le problème.

On définit alors l'ensemble des dernières positions potentielles d'une famille de fluents de position :

Définition 43 : Positions potentielles $\Omega(\Pi, t, \mathcal{P})$

Soit Π un plan partiel, t un instant temporel de ce plan et $\mathcal{P}$ une famille de fluents de position.

Soit $T_{\mathcal{P}}$ l'ensemble des tâches du plan créant des fluents de position de cette famille :

$$T_{\mathcal{P}} = \{\tau \in \mathcal{T}(\Pi) \mid \exists f \in Pre(\tau) \wedge f \in \mathcal{P}\}$$

Soit $T_{\mathcal{P},t}$ l'ensemble de ces tâches ayant un effet en t (elle doivent arriver avant t mais pas avant une autre tâche de mouvement) :

$$T_{\mathcal{P},t} = \left\{\tau \in T_{\mathcal{P}} \mid t^e_\tau \prec t \wedge \forall \tau_i \in T_{\mathcal{P}}, t \prec t^e_{\tau_i} \vee t^e_{\tau_i} \prec t^s_\tau\right\}$$

On peut alors en tirer l'ensemble des positions potentielles à un instant t :

$$\Omega(\Pi, t, \mathcal{P}) = \{f \in \mathcal{P} \mid \exists \tau \in T_{\mathcal{P},t}, f \in Add(\tau)\}$$

Si un plan ne possède aucune menace, alors toutes les actions de déplacement concernant une famille de fluents de position donnée sont complètement ordonnées. Dans ce cas, l'ensemble Ω ne contient qu'un seul élément : la dernière position connue à l'instant t.

On définit alors une nouvelle variante de h^{add} :

Définition 44 : h^{add}_{motion}

Soit Π un plan partiel et $OL(\Pi)$ la liste des liens ouverts de Π.

$$h^{add}_{motion}(\Pi) = \sum_{(t,f)\in OL(\Pi)} \begin{cases} \min\limits_{g\in\Omega(\Pi,t,\mathcal{P})} dur(a^{g\to f}) & \text{si } \exists \mathcal{P} \mid f \in \mathcal{P} \\ h^{add}(f) & \text{sinon} \end{cases}$$

Si plusieurs actions produisant des fluents de position ne sont pas complètement ordonnées, on considère la position la plus proche géographiquement (*i.e.* celle dont le coût est le plus faible) parmi celles qui pourraient apparaître en dernier dans le plan avant le lien ouvert considéré.

Cette amélioration de h^{add} peut être utilisée en même temps que les versions *reuse* et *areuse*. On utilise alors la version *motion* pour les fluents de position et la version *reuse* ou *areuse* pour les autres fluents. On peut donc avoir des configurations utilisant à la fois h^{add}_{motion} et h^{add}_{reuse}.

Exemple : Utilisation des fluents de position dans h^{add}

En reprenant l'étape présentée à la figure 4.2f et rappelée ci dessous, on peut mettre en évidence le changement dans l'évaluation d'un plan.

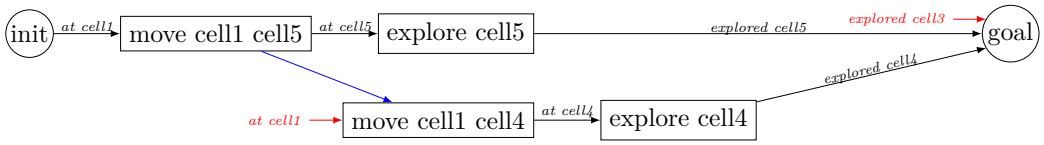


Figure 4.6 – Exemple de plan où les fluents de position sont utilisés dans le calcul de h^{add}. Dans ce cas de figure, sans l'utilisation des fluents de position, le lien ouvert vers `at cell1` est évalué à 0. h^{add}_{motion} l'évalue correctement à 4.

Le fluent `at cell1` étant vrai à l'état initial, son évaluation par h^{add}, h^{add}_{reuse} et h^{add}_{areuse} est 0. Ce plan parait donc très bon : le planificateur pense qu'il peut avoir le robot en `cell1` gratuitement. Les modifications présentées précédemment vont détecter dès l'introduction du lien causal entre `init` et `move cell1 cell4` que le plan est invalide. Mais d'ici là, l'évaluation de ce plan le sous-estime et pousse donc le planificateur à s'y intéresser.
La dernière action modifiant la position du robot avant le lien causal est `move cell1 cell5`. h^{add}_{motion} va donc évaluer le coût du lien ouvert vers `at cell1` à $dur(a^{cell5 \to cell1}) = 4$. Cette évaluation correspond effectivement à la durée la plus courte nécessaire pour passer de `cell5` à `cell1`. Elle va permettre au planificateur de détecter plus tôt que passer par `cell1` pour aller de `cell5` à `cell4` n'est pas très efficace et donc va lui permettre de ne pas explorer cette direction.

Le tableau 4.3 présente les résultats de la résolution de cet exemple avec les modifications proposées. On observe bien une amélioration des performances en termes de temps de calcul et de nombre de plans explorés. Comme l'heuristique s'est améliorée, elle a pu mieux guider la recherche. Par contre, le planificateur trouve encore un plan sous-optimal.

Moyenne sur 20 essais	Plans explorés	Durée de la recherche (ms)	Longueur du plan
No motion	80.00	93.30	9.01
Mutex	86.00	85.75	9.01
Mutex + h^{add}_{motion}	53.00	47.45	9.01

Tableau 4.3 – Résultats comparatifs sur un exemple en utilisant la configuration HiPOP-*local-areuse-earliest-time*.

4.3.2 Nouvel ordre de résolution des défauts

La sous-optimalité dans la résolution des problèmes provient de l'introduction précoce de liens causaux. Dans l'exemple présenté précédemment, le plan le plus court n'est pas trouvé : le planificateur trouve un plan sous-optimal. Au lieu de visiter les lieux dans l'ordre

optimal, il commence par aller directement au plus lointain, ignorant toutes les cellules sur son chemin, avant de revenir en arrière pour les explorer.

Cet effet a plusieurs causes, mais la principale concerne l'ajout trop précoce de liens causaux sur les positions. L'heuristique *local*, qui a prouvé son intérêt dans le cas général, pousse ici à s'assurer le plus tôt possible que la position nécessaire du robot pour l'observation est bien atteignable. Cela passe par l'ajout le plus tôt possible de l'action de déplacement correspondante et des liens causaux. Une fois ces liens dans le plan, il n'est plus possible de les enlever. On force donc le planificateur à ne pas changer d'avis et à aller directement de la `cell1` à la `cell5`.

Or les actions de déplacement garantissent qu'il sera toujours possible d'atteindre une position depuis n'importe quelle autre position. Il n'est donc pas nécessaire de résoudre les liens ouverts concernant les positions le plus tôt possible. Et sans les liens causaux, il devient possible pour le planificateur de changer l'ordre (temporel) de résolution des buts sans avoir à les résoudre dans l'ordre donné par l'heuristique de sélection des défauts.

On propose donc d'ordonner les défauts de manière à résoudre les liens ouverts concernant des fluents de position en dernier (parmi les liens ouverts).

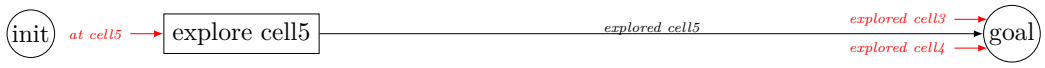
Exemple : Sélection des liens ouverts de position en dernier

Résoudre les liens ouverts concernant les positions à la fin change radicalement la recherche qui a été présentée jusqu'ici. Au début, le plan initial est le même.

Figure 4.7 – Exemple de résolution d'un problème en sélectionnant les fluents de position en dernier.

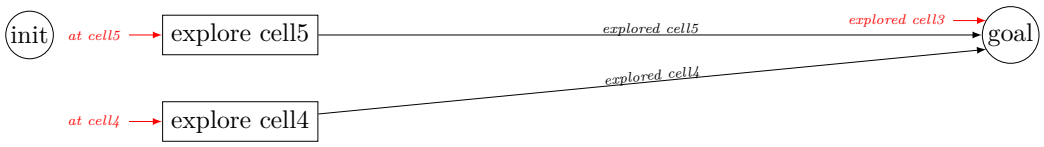
(a) Plan vide.

Il n'y a pas de lien ouvert concernant des positions. Donc le premier lien ouvert sélectionné est le même : celui dont l'évaluation est la plus grande.



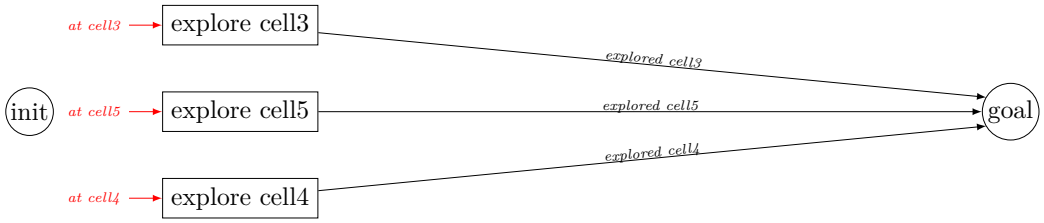
(b) Le premier but en cours de résolution : l'action d'exploration a été ajoutée. Il n'y a pas de changement notable.

Le nouveau lien ouvert introduit est une position. Il n'est donc pas résolu tout de suite (même si l'heuristique *local* l'imposerait normalement). A la place, on va résoudre le prochain lien causal : le plus coûteux parmi ceux restant.



(c) Bien que l'on utilise l'heuristique *local*, on résout les liens ouverts concernant les positions à la fin. Le planificateur a donc cherché à résoudre un autre lien ouvert, en l'occurrence celui sur `explored cell4`

Le même phénomène se répète et mène à un plan où tous les liens ouverts sont des liens de position.



(d) Tous les liens ouverts qui ne concernaient pas des positions ont été résolus.

Maintenant, l'ordre de résolution de ces liens ouverts va déterminer dans quel ordre les actions seront exécutées car les différentes actions d'observation n'ont aucune contrainte temporelle associée.

Ordonner les liens ouverts pour résoudre ceux portant sur les fluents de position en dernier permet de ne pas forcer le plan à commencer par le premier but de haut niveau choisi. Mais cela ne garantit pas qu'ils seront ensuite résolus dans le meilleur ordre car aucun mécanisme jusque-là ne permet de les différencier. Cela sera l'objet de la prochaine modification.

Le tableau 4.4 présente les résultats de la résolution de cet exemple avec les modifications proposées. Le fait de résoudre les liens ouverts de position en dernier est appelé *motionLast*. La longueur du plan trouvé est meilleure, mais elle est encore sous-optimale. Et le nombre de plans explorés ainsi que le temps de calcul a augmenté.

Moyenne sur 20 essais	Plans explorés	Durée de la recherche (ms)	Longueur du plan
No motion	80.00	93.30	9.01
Mutex	86.00	85.75	9.01
Mutex + h^{add}_{motion}	53.00	47.45	9.01
Mutex + h^{add}_{motion} + motionLast	61.00	67.75	8.01

Tableau 4.4 – Résultats comparatifs sur un exemple en utilisant la configuration HiPOP-*local-areuse-earliest-time*.

4.3.3 Ajout automatique de contraintes temporelles

Pour résoudre les actions dans le bon ordre, il faut un moyen de les distinguer les unes des autres pour savoir lesquelles ordonnancer en premier. Or nous avons déjà fait une modification qui permet de les différencier en utilisant la dernière position connue du robot avant un lien ouvert dans le calcul de h^{add}_{motion}. Si le but est d'optimiser la durée du plan, ce coût est la durée minimale nécessaire pour passer de la position précédente à la position voulue. Le fait que les actions de déplacement représentent le moyen le plus efficace d'aller d'un point à un autre garantit que ce coût est la durée minimum qui doit exister entre ces deux positions dans tous les plans, même si le planificateur décide de prendre un autre chemin pour y aller. On se propose donc ici d'ajouter directement dans le plan cette contrainte temporelle. Donc pour chaque lien ouvert, on cherche la dernière position garantie du robot et on ajoute dans le plan un contrainte de séparation temporelle minimale entre ces deux points. Ces contraintes, différentes pour chaque lien ouvert, vont permettre de distinguer les actions.

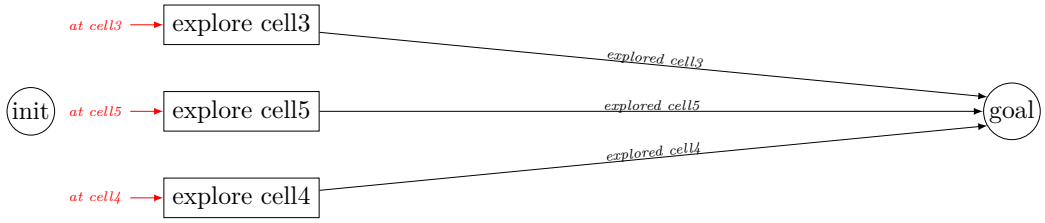
Le STN utilisé pour gérer les contraintes temporelles (cf. définition 25 page 33) permet d'ajouter des contraintes de séparation temporelle au même titre que des contraintes de précédence, sans différence dans leur traitement. De plus, même si le planificateur décide ensuite de passer par une position intermédiaire, cette contrainte restera valide.

Une fois cet ordre général déterminé, il est préférable d'établir les liens ouverts dans l'ordre temporel. Ainsi on peut utiliser les actions de déplacement passées pour établir les liens causaux nécessaires pour les positions futures. On se propose donc de résoudre les positions dans l'ordre naturel : en commençant par l'état initial et dans l'ordre du plan. C'est ce qui correspond à l'heuristique *earliest* qui garantit que parmi tous les liens ouverts de position (gardés pour la fin) on sélectionnera en priorité les liens ouverts arrivant le plus tôt dans le plan.

Exemple : Ajout de contraintes temporelles

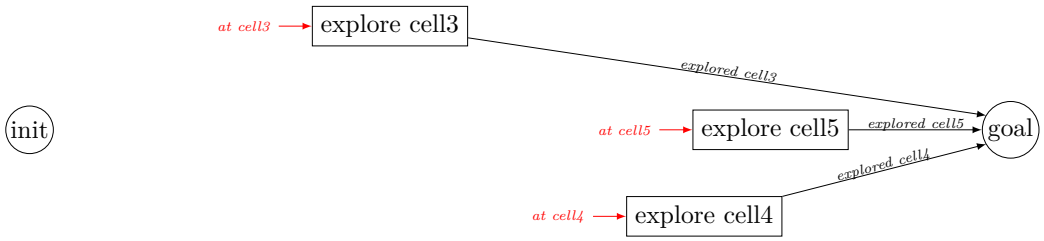
En reprenant l'étape présentée à la figure 4.7d et rappelée ci-dessous, on peut mettre en évidence la différenciation entre chaque action.

Figure 4.8 – Exemple de résolution d'un problème en ajoutant de nouvelles contraintes temporelles.



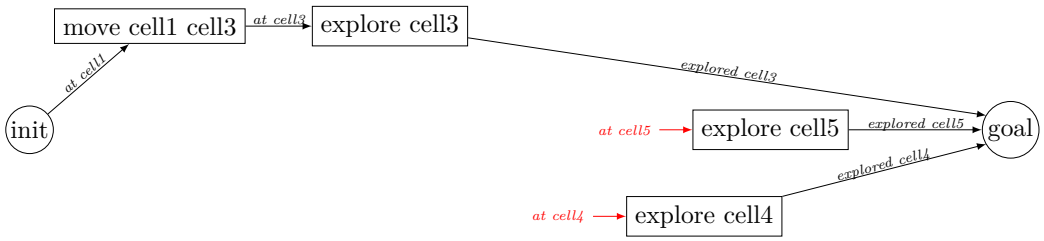
(a) Plan présentant des liens ouverts concernant uniquement des liens ouverts de position.

Les trois liens ouverts présents sont des liens ouverts de position. La dernière position garantie du robot est celle de l'état initial. Pour chaque action d'exploration, on peut donc ajouter une contrainte temporelle. Pour ces trois actions, la date de début minimale est (respectivement) de 2, 3 et 4. On obtient alors le plan suivant :



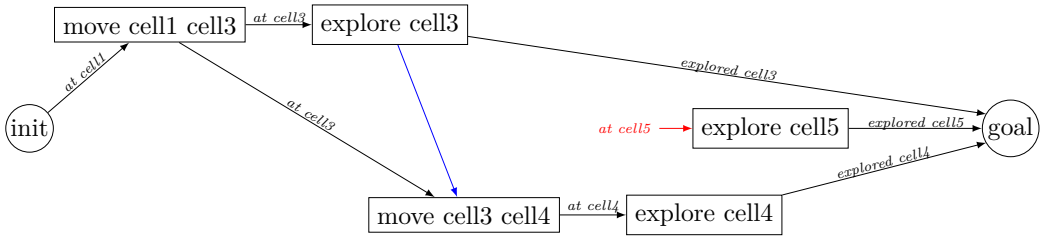
(b) Plan produit une fois que les nouvelles contraintes temporelles sont ajoutées. Ces contraintes sont représentés par les décalages relatifs entre les actions. La distance entre l'action initiale et `explore cell3` est de 2. Elle est de 1 entre `explore cell3` et `explore cell4` et entre `explore cell4` et `explore cell5`.

L'heuristique *earliest* va sélectionner en premier le lien ouvert vers `at cell3`. Sa résolution mène au plan suivant :



(c) Plan produit après la résolution du premier lien ouvert de position.

Les nouvelles contraintes temporelles qui devraient être ajoutées ici (maintenant que la dernière position connue a changé pour les deux derniers liens ouverts) ne changent pas le plan. Elles sont déjà prises en compte dans les contraintes temporelles qui sont dans le plan. La résolution du prochain lien causal donne ce plan :



(d) Plan produit après la résolution des deux premiers liens ouverts de position.

L'action de déplacement menace le lien causal sur `at cell3`. Le planificateur a donc ajouté une contrainte de précédence (en bleu) pour l'éviter. La résolution du dernier lien ouvert mène alors au plan optimal.

Le tableau 4.5 présente les résultats de la résolution de cet exemple avec les modifications proposées. Le fait d'introduire ces menaces nécessitant une séparation temporelle est appelée *temporalThreats*. HiPOP trouve donc un plan optimal en explorant moins de plans (et plus

rapidement) que les autres configurations.

Moyenne sur 20 essais	Plans explorés	Durée de la recherche (ms)	Longueur du plan
No motion	80.00	93.30	9.01
Mutex	86.00	85.75	9.01
Mutex + h^{add}_{motion}	53.00	47.45	9.01
Mutex + h^{add}_{motion} + motionLast	61.00	67.75	8.01
Mutex + h^{add}_{motion} + motionLast + temporalThreats	23.00	33.05	7.01

Tableau 4.5 – Résultats comparatifs sur un exemple en utilisant la configuration HiPOP-*local-areuse-earliest-time*.

4.3.4 Adaptation de h^{add} à l'ajout automatique de contraintes temporelles

Une fois ces contraintes temporelles ajoutées, la question se pose de l'interaction avec h^{add}_{motion}. En effet l'idée de h^{add}_{motion} est d'utiliser les coûts comme une évaluation de la difficulté d'atteindre une position donnée. Alors qu'avec la modification précédente, cette information est directement intégrée dans le plan.

L'hypothèse de h^{add}_{motion} d'indépendance des liens causaux n'est pas vérifiée dans le cas où plusieurs robots sont présents par exemple : les déplacement d'un des robots peuvent être concurrents avec les déplacements d'un autre robot. Mais même dans le cas où un seul robot est présent (comme on peut le voir sur la figure 4.8c), plusieurs liens ouverts de position peuvent interagir ensemble.

Par contre l'ajout dans le plan des contraintes temporelles permet de prendre en compte ces interactions plus finement : les interactions sont correctement prises en compte et la durée du plan augmente du minimum possible.

On se propose donc de changer le calcul de h^{add}_{motion} pour ne pas prendre en compte les liens ouverts vers les fluents de position. Cela nécessite d'avoir introduit les contraintes temporelles mentionnées à la section précédente mais permet à l'heuristique d'évaluer de manière plus fine le coût d'un plan. Cette variante de h^{add} est appelée h^{add}_{ncm} pour *no cost motion*.

Définition 45 : h^{add}_{ncm}

Soit Π un plan partiel et $OL(\Pi)$ la liste des liens ouverts de Π.

$$h^{add}_{ncm}(\Pi) = \sum_{(t,f)\in OL(\Pi)} \begin{cases} 0 & \text{si } \exists \mathcal{P} \mid f \in \mathcal{P} \\ h^{add}(f) & \text{sinon} \end{cases}$$

Cette modification a cependant un désavantage : elle crée des plateaux dans l'heuristique. En effet, l'introduction de liens ouverts de position pour un robot qui n'est pas celui qui définit la longueur du plan (*i.e.* celui qui n'est pas le dernier à finir sa partie du plan) n'ajoutera aucun coût : son évaluation par h^{add} vaudra 0 et la durée du plan ne changera

pas. Et si plusieurs chemins existent pour se rendre d'une position à une autre, tous les plans empruntant ces chemins auront le même coût.

Cela va créer un ensemble de plans ayant un coût équivalent : si on a déjà prévu le temps nécessaire aux déplacements dans le plan (avec les menaces temporelles), tous les plans correspondant aux différents chemins possibles pour aller d'un point à un autre sont équivalents. Si l'évaluation heuristique d'un plan sous-estime son coût, le planificateur va chercher à vérifier tous les plans ayant ce coût inférieur avant de considérer un plan ayant un coût supérieur (*i.e.* il va devoir explorer un « plateau »). Pour limiter la taille de ces plateaux, on utilise un mécanisme proche de celui utilisé lors de l'instanciation des actions hiérarchiques. La propriété d'existence des actions de déplacement pour tout couple de points de départ et d'arrivée garantit que quand il ne reste plus que des liens ouverts de position à résoudre (car ils sont résolus en dernier parmi les liens ouverts) une solution sera forcément trouvée. Dans ce cas, il n'est pas nécessaire de garder en mémoire et d'explorer les alternatives possibles. Si on ne faisait pas cela, lors de l'instanciation des actions abstraites (qui vient après celle des liens ouverts), une mauvaise estimation heuristique d'une action hiérarchique conduirait à explorer tous les chemins possibles pour tous les liens ouverts de position qui ont été résolus jusque-là avant d'accepter de rechercher parmi des plans avec une évaluation heuristique plus importante.

Le tableau 4.6 présente les résultats de la résolution de cet exemple avec les modifications proposées. Le nombre de plans explorés augmente légèrement mais la durée de la recherche diminue. On trouve toujours le plan optimal : cet exemple ne permet pas de vérifier si cette modification va bien engendrer des plans plus courts.

Moyenne sur 20 essais	Plans explorés	Durée de la recherche (ms)	Longueur du plan
No motion	80.00	93.30	9.01
Mutex	86.00	85.75	9.01
Mutex + h^{add}_{motion}	53.00	47.45	9.01
Mutex + h^{add}_{motion} + motionLast	61.00	67.75	8.01
Mutex + h^{add}_{motion} + motionLast + temporalThreats	23.00	33.05	7.01
Mutex + h^{add}_{ncm} + motionLast + temporalThreats	26.00	21.70	7.01

Tableau 4.6 – Résultats comparatifs sur un exemple en utilisant la configuration HiPOP-*local-areuse-earliest-time*.

4.4 Validation expérimentale de ces modifications

Sur l'exemple présenté jusqu'ici, on note une amélioration certaine des performances de HiPOP par l'utilisation d'un raisonnement spécifique sur les fluents de position. Nous avons donc repris les évaluations faites dans le chapitre 3 pour étudier l'influence de ces modifications sur différents domaines.

Deux heuristiques utilisant un raisonnement géométrique sont proposées : une version utilisant toutes les modifications présentées dans ce chapitre et une version utilisant h^{add}_{motion} plutôt que h^{add}_{ncm}. Ces heuristiques sont respecivement appelée *motion* et *ncm*. Elles

impliquent l'utilisation de *earliest*. Ces configurations correspondent aux deux dernières lignes du tableau 4.6.

Dans les domaines étudiés, deux domaines présentent des familles de fluents de position qui ont été détectés par HiPOP :

- *satellite* : la direction dans laquelle pointe le satellite `pointing ?s ?d` est détectée comme un fluent de position.
- *survivors* : les fluents décrivant la position des robots `at ?r ?l` et des équipes `at-team ?t ?z` sont correctement détectés. Les positions des blessés `at-s ?s ?l` ne sont pas considérées comme des fluents de position car l'action de déplacement n'existe pas : ils ont besoin des robots pour se déplacer.

Le domaine *blocks* ne présente pas de fluents de position tels que nous les avons définit. La modélisation des autres domaines (*logistics* et *gripper*) n'a pas permis de détecter les fluents de position par l'algorithme de détection implémenté par HiPOP. En particulier dans *logistics* c'est le même prédicat qui est utilisé pour la position des paquets et des véhicules. Or la position d'un paquet peut « disparaitre » s'il est chargé dans un véhicule : plus aucun fluent de prédicat `at` n'est vrai, à la place on a un fluent de prédicat `in`. Ainsi, même si des familles de fluents de position existent, comme ils ne partagent pas le même prédicat ils ne sont pas détectés. Les résultats présentés dans ce chapitre ne concernent donc que ces deux domaines, les résultats sur les autres domaines restent inchangés.

Nous allons chercher à répondre à plusieurs questions :

- **Est-ce que ces modifications sur les fluents de position améliorent les performances de HiPOP ?**
- **Quel est l'apport de l'utilisation des actions hiérarchiques pour la planification avec ces modifications ?**
- **Est-ce que les heuristiques *time* et *ncm* permettent d'efficacement optimiser la durée des plans trouvés avec ces modifications ?**
- **Comment est-ce que HiPOP avec ces modifications se compare avec d'autres planificateurs ?**

4.4.1 Résultats sur les benchmarks

Est-ce que ces modifications sur les fluents de position améliorent les performances de HiPOP ? La figure 4.9 présente les résultats de différentes configurations sur les deux domaines sélectionnés précédemment pour plusieurs configurations de HiPOP. On a choisi de ne représenter sur le courbes que les configurations utilisant l'heuristique *local* car toutes ces configurations utilisent *earliest* et la différence entre *local* et *sorted* est faible dans ce cas (ce que l'on a vérifié expérimentalement).

On voit ici très nettement une amélioration des performances de HiPOP dans ces domaines présentant des fluents de position : on arrive à résoudre tous les problèmes de *survivors* et de *satellite*. Dans *survivors*, on passe de quelques problèmes résolus (moins de 10) à 72 (c'est-à-dire tous). Cela confirme notre analyse : c'est le raisonnement géométrique qui faisait défaut à HiPOP. L'utilisation de h^{add}_{ncm} fait ici baisser le taux de résolution des problèmes, sans doute à cause de l'augmentation de la taille des plateaux.

Dans le domaine *satellite*, l'utilisation de h^{add}_{ncm} permet de résoudre tous les problèmes en moins de une minute là où les configurations n'utilisant pas de raisonnement géométrique

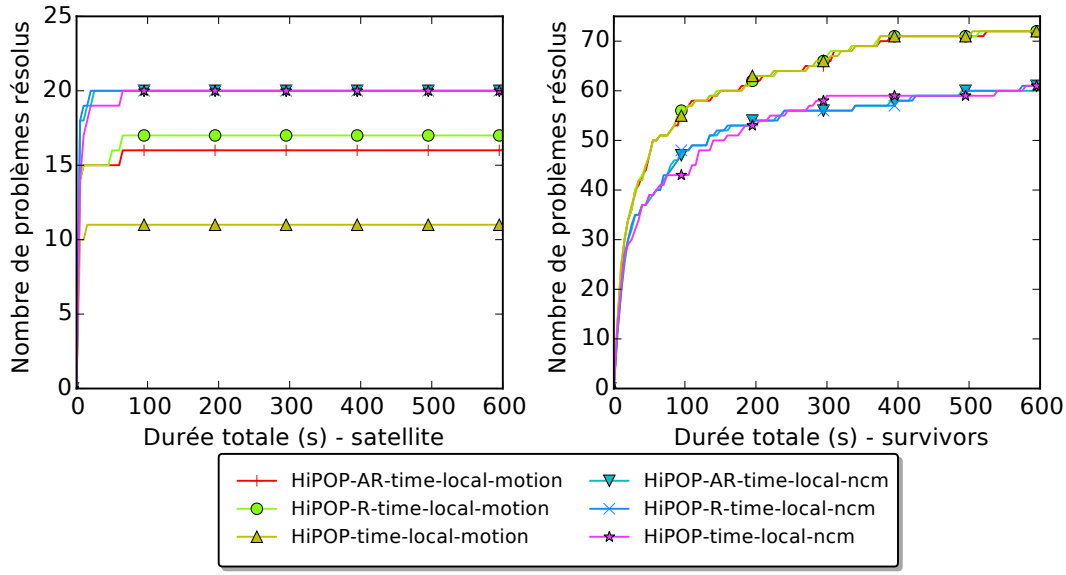


Figure 4.9 – Nombre de problèmes résolus par plusieurs configurations d'HiPOP utilisant *motion*. Comparaison des utilisations de h^{add}_{reuse}, h^{add}_{areuse}, h^{add}_{motion} et h^{add}_{ncm} en conjonction avec *time* et *local*. Chaque graphique correspond à un domaine différent et représente en ordonnée le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse.

ne les résolvaient pas tous. On observe alors le phénomène inverse par rapport à *survivors* : l'utilisation de *ncm* améliore le temps de calcul et donc le nombre de problèmes résolus.

Quel est l'apport de l'utilisation des actions hiérarchiques pour la planification avec ces modifications ? La figure 4.10 présente les résultats des meilleures configurations utilisant l'heuristique *motion* utilisant ou non l'heuristique *time* et les actions hiérarchiques.

Sur *satellite* la version sans action hiérarchique résout toujours tous les problèmes, le changement vient du fait que maintenant la version les utilisant résout aussi tous les problèmes. Sur *survivors*, le raisonnement géométrique a permis la résolution de certains problèmes par une configuration temporelle *Bare*, ce qui n'était pas le cas auparavant. Néanmoins, le nombre de problèmes résolus est bien plus faible qu'avec l'utilisation des actions hiérarchiques.

On peut donc conclure que dans *survivors*, les actions hiérarchiques sont nécessaires pour résoudre tous les problèmes et que seul le domaine *satellite* présente de meilleurs résultats sans l'utilisation des actions hiérarchiques. La conclusion du chapitre précédent est donc toujours valide : il y a un apport net des actions hiérarchiques dans les domaines qui s'y prêtent.

Est-ce que les heuristiques *time* et *ncm* permettent d'efficacement optimiser la durée des plans trouvés avec ces modifications ? La figure 4.10 présente aussi la meilleure configuration non temporelle utilisant le raisonnement géométrique. Sur les deux domaines étudiés, le raisonnement non temporel est plus rapide (et dans *satellite* permet de résoudre plus de problèmes). La figure 4.11 présente la durée des plans produits

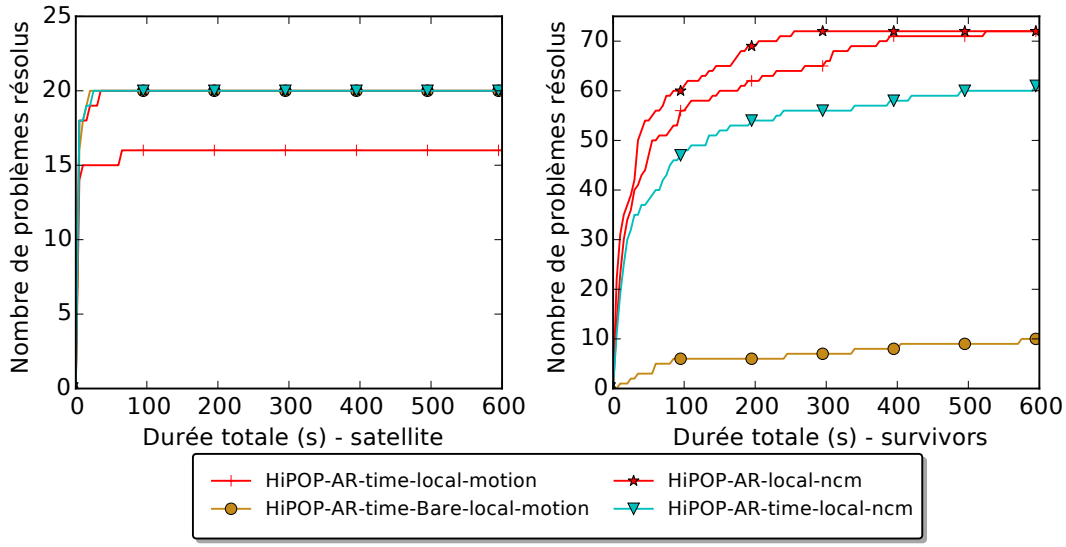


Figure 4.10 – Nombre de problèmes résolus par plusieurs configurations d'HiPOP utilisant *motion* avec ou sans les actions hiérarchiques. Comparaison des utilisations (ou non) de *time* et des actions hiérarchiques. Chaque graphique correspond à un domaine différent et représente en ordonnée le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse.

par certaines configurations utilisant *ncm* en fonction de la durée des plans trouvés par la configuration HiPOP-AR-time-local-motion. On a choisi de représenter la meilleure configuration *ncm* utilisant *time* et la meilleure configuration n'utilisant pas *time*.

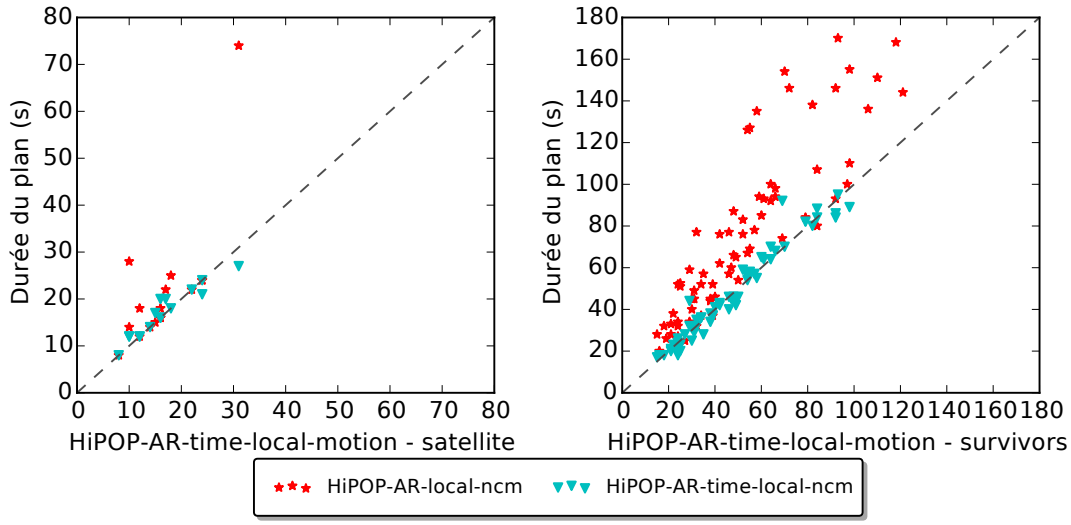


Figure 4.11 – Comparaison de l'utilisation ou non de l'heuristique *time* sur la qualité des plans trouvés par HiPOP. Chaque graphique correspond à un domaine différent et représente en ordonnée la taille du plan trouvé par la meilleure heuristique non temporelle en fonction de la taille du plan trouvé par la meilleure heuristique temporelle. Un point au-dessus de la première bissectrice signifie donc un plan plus court produit par la configuration utilisant *time* et pas *ncm*.

En ce qui concerne la configuration n'utilisant pas *time*, on observe que tous les points sont au-dessus de la bissectrice. Cela montre bien que l'heuristique *time* est adaptée au traitement des problèmes dont l'objectif est de minimiser la taille du plan. Cela induit une recherche plus complexe (ce qui se traduit par un temps de calcul plus long, pouvant éventuellement mener à l'échec de la résolution d'un problème) mais produit des plans ayant une durée plus courte.

Par contre la configuration utilisant *ncm* ne semble pas toujours donner de meilleurs plans. La répartition des points semble relativement équilibrée de chaque coté de la bissectrice. Néanmoins cette configuration résout moins de problèmes que celle n'utilisant pas *ncm* sur *survivors*.

Comment est-ce que HiPOP avec ces modifications se compare avec d'autres planificateurs ? La figure 4.12 présente le nombre de problèmes résolus en un temps donné pour plusieurs configurations d'HiPOP et pour les autres planificateurs TFD, YAHSP et VHPOP (présentés au chapitre 3). La figure 4.13 présente la taille du plan produit pour chaque problème. Une courbe au-dessus d'une autre signale donc un planificateur qui a trouvé un plan plus long pour un problème donné. Au vu des différences observées, l'axe des abscisses est gradué en échelle logarithmique.

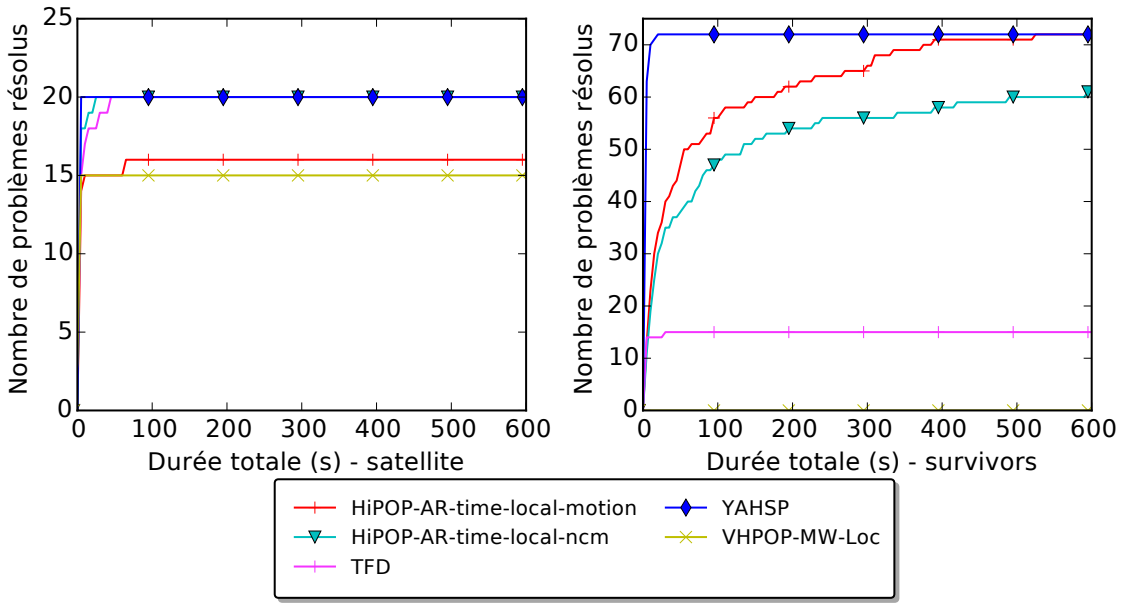


Figure 4.12 – Nombre de problèmes résolus par plusieurs configurations d'HiPOP utilisant *motion* ainsi que par plusieurs autres planificateurs. Comparaison de HiPOP avec d'autres planificateurs temporels. Chaque graphique correspond à un domaine différent et représente en ordonnée le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse.

On observe ici que les modifications effectuées n'ont pas radicalement changé le comportement d'HiPOP. HiPOP est toujours moins rapide que YASHP pour résoudre des problèmes, même sur les domaines présentant des fluents de position. Les plans trouvés par HiPOP sont généralement un ordre de grandeur plus court que ceux trouvés par YAHSP, même en mode *anytime*, et un ordre de grandeur moins bon que ceux trouvés par d'autres planificateurs temporels comme TFD. Cette différence s'explique en partie par le fait

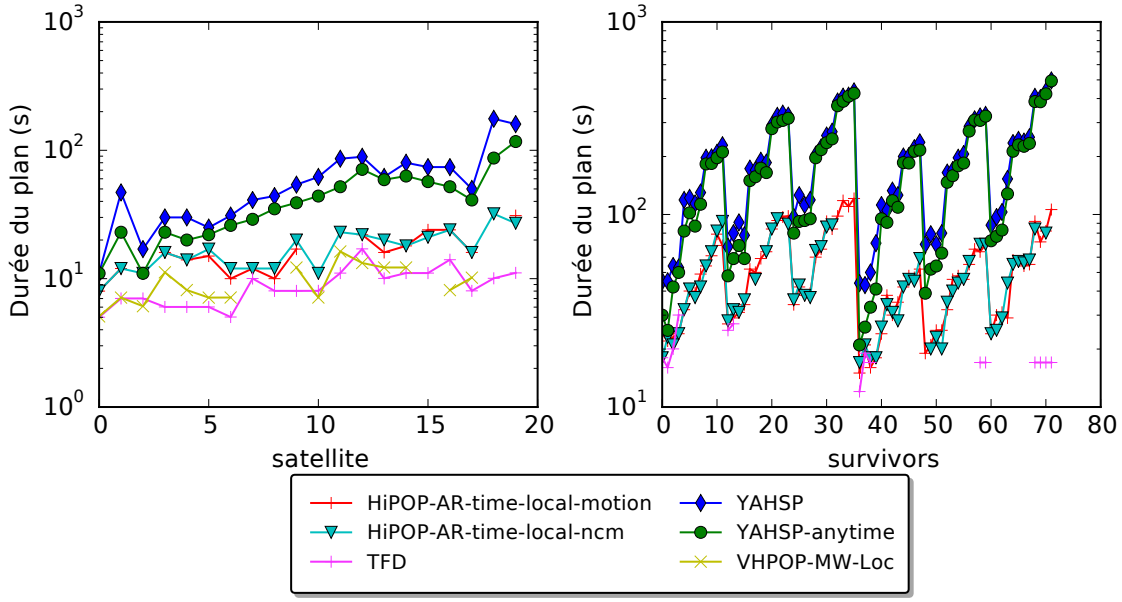


Figure 4.13 – Qualité des plans trouvés par plusieurs planificateurs sur différents domaines. Comparaison des meilleures configuration d'HiPOP avec les autres planificateurs en terme de durée du plan trouvé.

que HiPOP est contraint par la description hiérarchique fournie : il cherche uniquement des plans s'y conformant. Même quand il renvoie des plans optimaux dans son espace de recherche, ces plans peuvent être loin des plans optimaux accessibles pour les autres planificateurs.

Conclusion

Dans cette partie, nous avons vu comment utiliser les spécificités de certains fluents pour améliorer la recherche dans un planificateur symbolique.

Plus précisément, nous avons défini des familles de fluents de position comme étant une famille telle qu'un seul fluent est vrai à tout moment et telle qu'il existe des « actions de déplacement ». Ces actions doivent permettre de passer d'un fluent à l'autre au sein de cette famille dans un temps minimal au vu des autres actions du domaine et sans dépendre du reste de l'état courant.

Une fois ces familles identifiées, leurs propriétés peuvent être utilisées pendant la recherche. En particulier, elles permettent :

- de détecter au plus tôt des mutex pour améliorer la détection des menaces (cf. section 4.2) ;
- de prendre en compte le plan partiel en cours d'évaluation dans h^{add} et non pas uniquement l'état initial (cf. section 4.3.1) ;
- de proposer un nouvel ordre de résolution des liens ouverts pour permettre plus de flexibilité dans les plans trouvés (cf. section 4.3.2) ;
- d'ajouter automatiquement des contraintes temporelles dans le plan pour en améliorer l'estimation du coût (cf. section 4.3.3) ;

– de proposer une nouvelle modification de h^{add} pour adapter la recherche à ces nouvelles contraintes temporelles (cf. section 4.3.4).

Les résultats montrent que sur les domaines où des familles de fluents de position sont détectées, les gains sont très importants. En particulier, les problèmes apparus dans le chapitre précédent sur le domaine *survivors* sont résolus : HiPOP parvient à trouver un plan dans tous les problèmes du domaine.

L'algorithme proposé est alors en mesure de résoudre des problèmes multiagents afin de calculer un plan initial avant le début d'une mission. Il satisfait donc les contraintes qui ont été posées au début du chapitre 3.

– Il est capable de résoudre une variété de problèmes pour être le plus versatile possible et donc s'adapter à des évolutions du problème à résoudre. Il est ainsi capable de résoudre des problèmes dans cinq domaines différents présentant tous des caractéristiques différentes.

– Il est capable de trouver rapidement un plan initial pour un problème type d'exploration d'une zone par une équipe de robots. Dans les tests qui ont été présentés, le plus gros plan contient 300 actions pour 3 équipes de robots et est calculé en moins de 10 minutes.

– Il est capable de gérer des échéances. Cette capacité n'a pas été testée expérimentalement lors des benchmarks présentés mais a pu être testée sur des exemples construits spécifiquement (non présentés dans ce mémoire).

– Il produit un plan flexible temporellement, donc plus facilement exécutable.

Mais l'algorithme de planification doit aussi être capable de réparer un plan en cours d'exécution. L'adaptation de HiPOP pour permettre de réparer un plan est donc l'objet du chapitre suivant.

5

Adaptation d'un algorithme hybride à la réparation de plans

Sommaire

Le but de ce chapitre est d'adapter un algorithme de planification hybride (donc basé sur une recherche en avant) pour réparer un plan, *i.e.* résoudre un problème de planification en utilisant un plan en cours d'exécution, solution d'un problème proche, pour améliorer la recherche. On cherche à résoudre plus vite le problème qui se pose en utilisant le plan fourni, mais aussi à modifier le moins possible le plan actuellement en cours d'exécution et à garder impérativement les actions qui ne sont plus contrôlables. Cela permet aux robots d'avoir un comportement plus prédictible par l'opérateur humain qui les supervise et donc de faciliter cette supervision. Les plans produits par un algorithme de planification hybride contiennent aussi une hiérarchie parmi les actions. Nous allons aussi voir comment cette hiérarchie peut être utilisée par l'algorithme de réparation et si cette information supplémentaire peut être utilisée pour améliorer la réparation.

Les objectifs pour cet algorithme de réparation sont donc les suivants :

- Il doit pouvoir manipuler des plans flexibles temporellement, à moitié exécutés, contenant des échéances et une hiérarchie parmi les actions.
- Il doit être capable de réparer des plans dans notre domaine multirobot en un temps raisonnable.

– Il doit être robuste à la perte de communications entre les robots pendant l'exécution et à l'indisponibilité de certains robots

5.1 Choix de la technique de réparation

Comme pour la conception d'un algorithme de planification, la première étape dans la conception d'un algorithme de réparation consiste à identifier la technique la plus adaptée parmi les six présentées au chapitre 2.

5.1.1 Identification de la technique de réparation

La section 2.3 a présenté deux techniques permettant de limiter le besoin de réparation : la planification conditionnelle (cf. section 2.3.1) et la planification probabiliste (cf. section 2.3.2). Ces deux techniques nécessitent de produire des plans plus flexibles que ceux produits jusque-là par HiPOP. Nous avons décidé de ne pas modifier la forme des plans produits : la flexibilité temporelle permet déjà de limiter fortement le besoin de réparation dans le cas d'une action prenant du retard. Dans les autres cas, le nombre important d'aléas pouvant survenir nous a fait opter pour une réparation, ce qui n'implique pas d'avoir à raisonner sur tous les futurs possibles.

La section 2.2 a présenté quatre techniques de réparation :

- la recherche locale (cf. section 2.2.1) ;
- le déraffinement puis raffinement (cf. section 2.2.2) ;
- l'utilisation de règles spécifiques (cf. section 2.2.3) ;
- le rejeu du raisonnement (cf. section 2.2.4).

Les communications non garanties peuvent créer des situations où un groupe de robots perd le contact avec le reste de l'équipe. Chaque groupe peut donc réparer de son coté et chaque groupe peut donc être en train d'exécuter des plans issus de plusieurs réparations différentes au moment où la communication est rétablie. Il est donc nécessaire de pouvoir fusionner des plans issus de plusieurs réparations. Cette fusion ne permet pas de garder facilement le raisonnement qui a conduit à chaque plan au sein du plan fusionné : il y a en fait plusieurs raisonnements qui ont mené au plan en cours d'exécution. Nous avons donc éliminé la technique qui consiste à rejouer le raisonnement de la planification.

Notre algorithme de planification utilise déjà des connaissances supplémentaires, sous la forme d'actions hiérarchiques. On aurait alors pu en rajouter pour la réparation, par exemple sous la forme d'une action abstraite à introduire lors de la survenue d'un aléa. Les méthodes de cette action auraient été les différents moyens de surmonter cet aléa. Mais nous avons voulu limiter la quantité d'information supplémentaire qu'il est nécessaire de donner à HiPOP pour accomplir une réparation. On a déjà vu que les actions hiérarchiques ne sont pas nécessaires au fonctionnement de HiPOP, mais elles permettent d'améliorer ses performances. Pour la réparation, nous avons voulu réutiliser ces informations sans avoir à en fournir de nouvelles. De plus, fournir une action abstraite pour chaque aléa possible aurait limité les types d'aléas qui peuvent être traités par une réparation. Lors de la planification initiale, la mission est mieux formalisée et une description hiérarchique est moins gênante, mais lors de la réparation le but est notamment de réagir à certains événements qui n'ont pas été prévus. Il nous a donc semblé plus important de rester générique lors de la réparation. Nous avons donc éliminé l'utilisation de règles spécifiques.

Les deux choix qui s'offrent alors sont la recherche locale (qui permet d'ajouter ou d'enlever des éléments du plan pour se rapprocher petit à petit d'un plan solution) ou alors le retrait en une fois de plusieurs éléments du plan suivi par une recherche en avant

classique, sans retrait additionnel d'éléments. La recherche locale aurait nécessité plus de modifications : les heuristiques de la recherche auraient dû être adaptées au fait que des éléments du plan peuvent être retirés, un mécanisme aurait dû être mis en place pour garantir l'absence de boucles dans la recherche, etc. Au contraire, le retrait de tous les éléments en une seule fois, avant une recherche en avant, permet de garder sans la modifier toute la partie de recherche. Il faut alors adapter l'algorithme pour permettre de calculer (itérativement) un plan initial pour la recherche étant donné le plan en cours d'exécution et le problème à résoudre. Le reste de l'algorithme peut être réutilisé. C'est donc le choix que nous avons fait : la génération itérative d'un point de départ pour une planification hybride.

5.1.2 Identification des entrées d'un problème de réparation

La réparation peut partir de plusieurs postulats (cf. section 2.1.2) en ce qui concerne le lien entre le plan à réparer et le problème à résoudre.

Dans ce chapitre, on appellera *plan initial* d'un problème de réparation le plan en cours d'exécution et *problème initial* le problème dont le plan initial est solution. Si plusieurs réparations ont lieu successivement au cours de la mission, le plan solution d'une réparation est donc le plan initial de la suivante. L'occurrence d'un aléa correspond donc à la modification du problème initial pour créer un nouveau problème. Si le calcul du plan n'utilise pas le plan initial, on parle de replanification (on a en fait une planification classique : on doit trouver la solution d'un problème sans information supplémentaire). Ces relations sont représentées sur la figure 5.1.

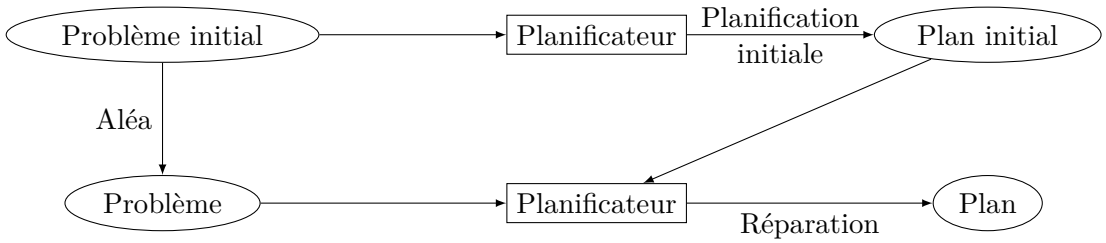


Figure 5.1 – Représentation des différents plans et problèmes intervenant dans la réparation.

Une partie des travaux existants considère qu'entre le problème initial et le problème de réparation, seul l'état initial ou les buts à atteindre changent. Cela permet de gérer certains cas d'aléa, mais cette hypothèse est quand même restrictive. L'hypothèse de base est que l'état initial du monde correspond à l'état actuel du monde. Au fur et à mesure que le temps passe, l'état initial est mis à jour. Les actions effectuées sont retirées du plan et leurs effets sont inclus dans le nouvel état initial. Cela permet donc de gérer le cas où une action échoue (par exemple elle n'effectue pas ses effets) mais pas celui où une action échoue de manière permanente. Par exemple si un nouvel obstacle est détecté, qui empêche tout mouvement passant par la position de l'obstacle, il est nécessaire de modifier les fluents statiques de la description du problème pour empêcher tout futur mouvement. On ne peut pas juste changer les fluents présents dans l'état initial du monde, les actions disponibles doivent aussi changer car les contraintes d'instanciation changent.

On se propose donc d'autoriser un changement complet du problème (en terme de description PDDL, cela revient à autoriser un changement du domaine ou du problème).

Ainsi le planificateur a comme entrée la description du problème et le plan initial. Les actions partageant le même nom dans le problème initial et dans le problème sont supposées avoir les mêmes description, mais des actions peuvent être ajoutées ou retirées dans le problème. Le planificateur peut déduire l'état initial et les buts du problème initial à partir du plan (en regardant les liens causaux depuis la tâche *Init* et vers la tâche *End*). Il n'a donc pas besoin de la description du problème initial.

En ce qui concerne l'exécution du plan, chaque tâche possède un état : elle peut être exécutée, en cours d'exécution ou planifiée. Le planificateur ne peut que modifier les tâches planifiées, il ne peut pas changer le passé. Une fois une tâche exécutée, sa date précise de début est connue (et sa date de fin est connue si l'exécution a fini). Les dates d'exécution de tous les instants temporels passés sont incluses dans le plan, et cela détermine l'état de chaque tâche.

En plus des dates de réalisation des tâches, la date courante est nécessaire. En effet, il n'est pas possible d'ajouter des actions dans le passé donc toute nouvelle action doit être ajoutée après la date courante.

La dernière information qui doit être incluse dans le plan concerne la présence des échéances. Pour chaque but qui doit être réalisé à une date donnée, une action but a été introduite et cette action a reçu une date précise, dans le futur. Cette date est donc incluse dans le plan. Le fait que cette date soit dans le futur permet de différentier les objectifs avec échéance des actions déjà exécutées.

Les entrées d'un problème de réparation, dans notre cas, comprendront donc :

- la description du domaine (au sens PDDL) ;
- la description du problème (au sens PDDL) ;
- la description des actions hiérarchiques du problème ;
- le plan initial en cours d'exécution, calculé sur un problème éventuellement différent du problème actuel. Ce plan contient, en plus des informations d'un plan non exécuté :
  - la date actuelle ;
  - un ensemble de dates correspondant aux instants temporels déjà exécutés ou aux échéances.

De plus, les robots pouvant perdre la communication avec le reste de l'équipe au cours de la mission, chaque robot doit être en mesure de réparer son plan seul. La réparation est donc embarquée à bord des robots.

5.2 Calcul du plan initial

Le processus de réparation utilisé repose donc sur la génération successive de plusieurs plans partiels, chacun utilisé comme point de départ pour une recherche hybride (cf. Algorithme 3 page 61). A chaque étape, on retire plus d'éléments du plan pour augmenter l'espace de recherche jusqu'à trouver une solution.

5.2.1 Algorithme de calcul itératif du plan de départ de la recherche

Le retrait itératif des tâches est inspiré de POPR [Krogt et Weerdt 2005] (cf. section 2.2.2). POPR utilise les liens causaux pour retirer les tâches qui sont proches du début ou

de la fin du plan. Nous avons décidé d'utiliser le même raisonnement pour retirer les tâches qui sont liées entre elles, mais en évaluant différemment le point de départ du retrait. En effet, si la réparation est nécessaire c'est qu'un problème existe dans le plan : des actions ne sont plus disponibles, une échéance n'est plus respectée, etc. Nous avons donc décidé de commencer le retrait des tâches en ces points où le plan semble poser problème. Si la réparation échoue, on retire alors localement plus de tâches en se concentrant sur les tâches qui sont liées localement aux tâches déjà retirées.

Le processus de génération de ces plans de départ de la recherche, à partir du plan initial, est présenté à la figure 5.2.

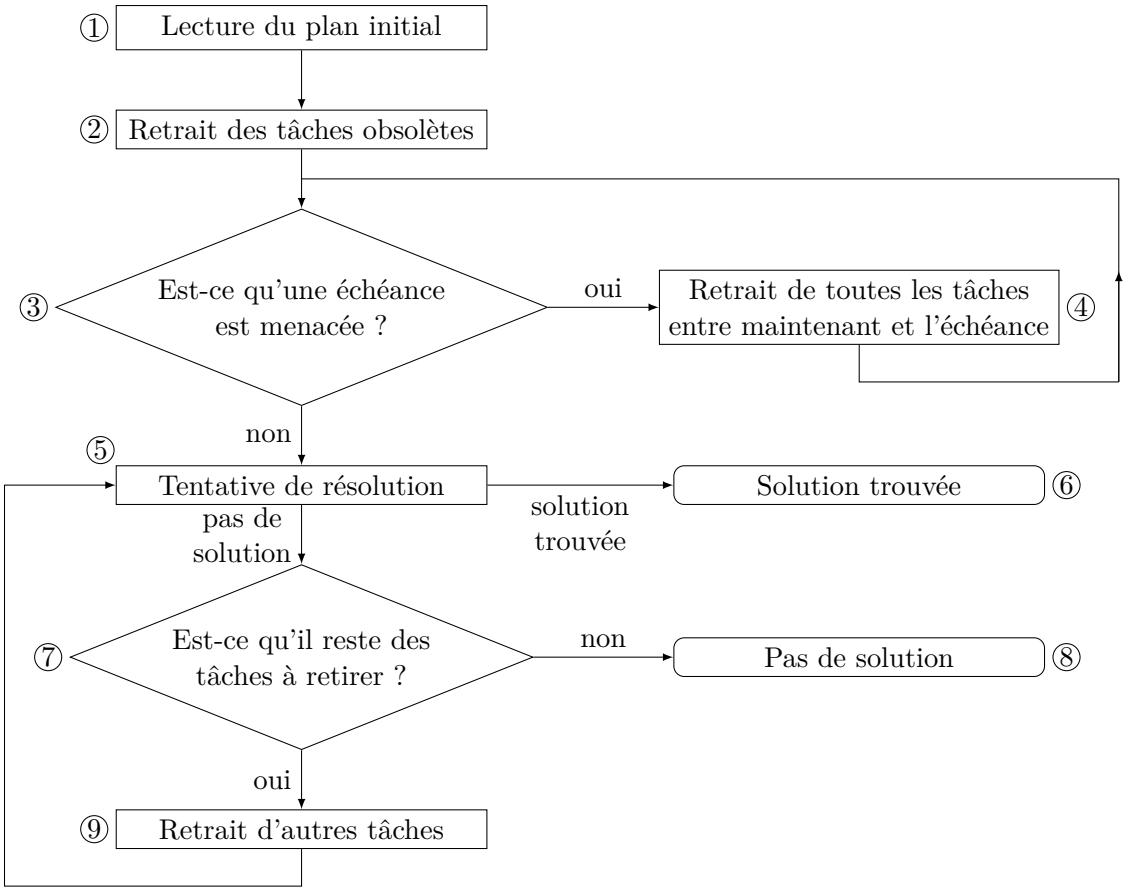


Figure 5.2 – Processus de réparation.

Les différentes étapes sont détaillées ci-dessous :

1. **Lecture du plan initial** : le plan à réparer est lu avec les nouvelles informations temporelles concernant les dates d'exécution de chaque tâche. Son STN associé est reconstruit.

2. **Retrait des tâches obsolètes** : on retire du plan actuel toutes les tâches futures qui ne sont plus disponibles dans ce domaine et toutes les tâches qui n'ont aucun lien causal sortant (et qui sont donc inutiles dans le plan). On répète ce processus jusqu'à ce que toutes les tâches du plan aient un lien causal sortant.

4. **Retrait des tâches précédant une échéance** : si une échéance n'est pas respectable dans le plan, c'est que toutes les actions la précédant ne sont pas compatibles

entre elles. Cela vient sans doute d'une tâche qui a pris du retard. Dans ce cas, on retire toutes les tâches précédant l'échéance menacée pour permettre de remettre à plus tard certaines actions afin de respecter l'échéance. Ce procédé est répété pour toutes les échéances menacées.

5. **Résolution** : on utilise le plan actuel comme point de départ pour une recherche hybride. Si une solution est trouvée, on a résolu le problème de planification. Sinon, il faut retirer de nouvelles tâches.

9. **Retrait des tâches suivantes** : si aucune solution n'a pu être trouvée, c'est sans doute que les tâches dans le plan actuel sont trop contraignantes. En partant des tâches qui ont été retirées jusque-là, on retire toutes les tâches qui leur sont liées causalement (*i.e.* toutes les tâches pour lesquelles il existe un lien causal entre elles et une tâche retirée). Si aucune nouvelle tâche n'est retirée et qu'il reste des tâches à retirer, alors on retire toutes les tâches du plan. On arrive alors à une replanification.

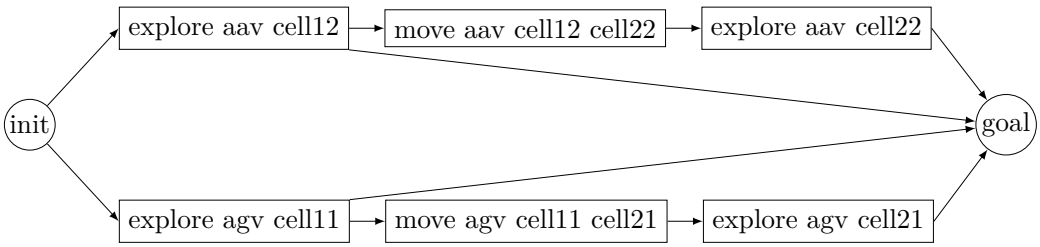
Dans tous les cas, le retrait de tâche n'est possible que si la tâche n'est pas une tâche passée (*i.e.* exécutée ou en cours d'exécution) et si la tâche n'est pas associée à une échéance (dans ce cas c'est une tâche but qui doit être dans le plan solution).

Ce processus est illustré sur l'exemple suivant :

Exemple : Retrait itératif d'éléments d'un plan élémentaire

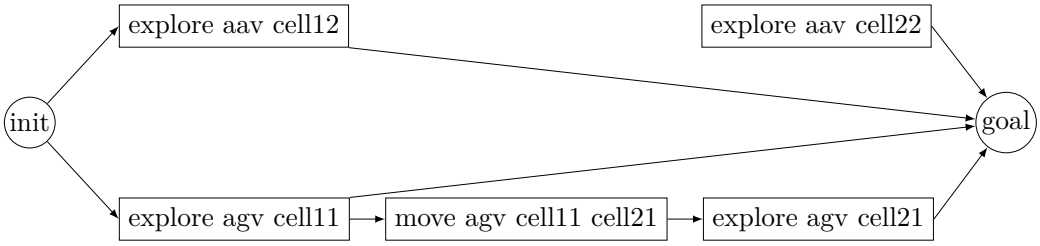
Considérons le plan élémentaire suivant :

Figure 5.3 – Exemple de retrait itératif d'éléments d'un plan élémentaire.



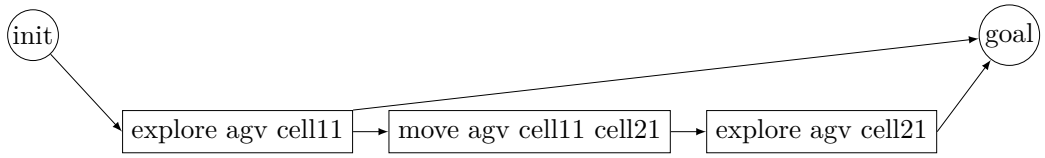
(a) Représentation du plan initial.

Supposons maintenant que l'action `move aav cell12 cell22` soit indisponible. Elle est retirée du plan et le plan initial pour la recherche est :



(b) Plan initial de la recherche si `move aav cell12 cell22` est indisponible.

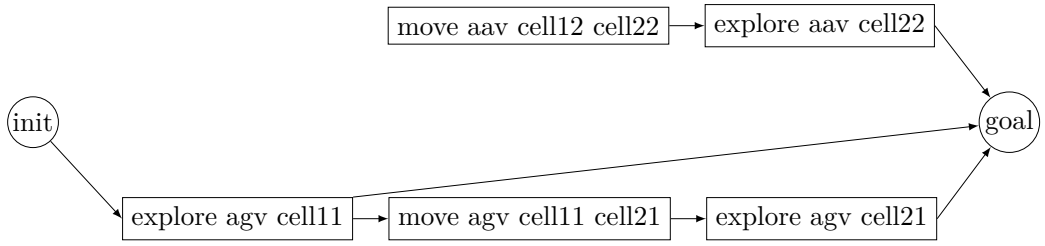
Si aucune solution n'est trouvée, on retire toutes les tâches potentielles qui lui sont reliées causalement. On aboutit alors au plan partiel suivant (où il ne reste plus que les actions du robot terrestre) :



(c) Plan initial de la recherche si `move aav cell12 cell22` est indisponible et que la première recherche a échoué.

Si aucune solution n'est trouvée dans ce cas de figure, on retire toutes les tâches restantes et on essaye de replanifier.

Supposons maintenant que l'on souhaite réparer ce plan en enlevant du but `explored cell12`. On retire alors du plan le lien causal établissant ce fluent et ensuite toutes les tâches n'ayant pas de lien causal sortant. On aboutit alors au plan initial suivant :



(d) Plan initial de la recherche si `explored cell12` est retiré du but.

Contraintes temporelles fantômes. Quand une tâche est retirée du plan, les liens causaux qui lui sont associés sont aussi retirés. En effet, ses préconditions n'ont pas à être assurées et ses effets ne sont plus assurés. Néanmoins, les liens causaux induisent aussi des contraintes temporelles entre les tâches. Et ces contraintes temporelles peuvent empêcher certaines menaces d'apparaître.

Par exemple lorsqu'un robot enchaîne les déplacements, les actions de déplacement sont ordonnées par les liens causaux : une action ne peut pas avoir lieu avant une autre car elle a besoin que la position du robot soit garantie avant de pouvoir s'exécuter. Mais si on retire un seul maillon de cette chaîne, alors il n'y a plus de contrainte temporelle entre deux chaînes de déplacement. Cela introduit un certain nombre de menaces entre différentes actions de déplacement. De plus, si le planificateur résout ces menaces en inversant l'ordre des actions, le plan fourni aura radicalement changé par rapport au plan initial, et cela sans raison apparente.

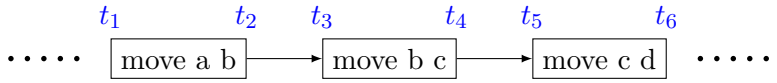
Pour garder une certaine stabilité du plan et pour limiter le nombre de menaces qui sont produites par le retrait des tâches, on introduit le concept de *liens fantômes*.

Chaque instant temporel retiré (en même temps que la tâche qui lui est associée) est gardé sous forme d'un instant temporel fantôme. Chaque lien causal ou lien temporel qui doit être retiré du plan est à la place gardé dans une liste sous forme de lien fantôme. Les tâches retirées définissent aussi des liens fantômes : une précédence entre l'ancien instant temporel de début et celui de fin. Une fois le plan construit et avant son utilisation, on introduit l'ensemble de toutes les contraintes temporelles qui seraient introduites si les instants temporels fantômes et les liens fantômes étaient introduits. Une fois le plan construit, les instants temporels fantômes et les liens fantômes ne sont plus utilisés. Mais ils permettent de maintenir les contraintes temporelles implicites dans le plan.

Exemple : Utilisation des contraintes fantômes

Considérons un plan contenant une suite d'actions de déplacement :

Figure 5.4 – Exemple d'utilisation des contraintes fantômes.

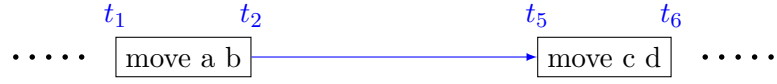


(a) Extrait d'un plan contenant une chaîne de trois actions de déplacement. Les instants temporels de chacune des tâches ont été notés en bleu au-dessus (un pour le début et un pour la fin de chaque action). Les liens causaux sont représenté en noir.

Supposons maintenant que la tâche `move b c` ne soit plus disponible (par exemple si on a repéré un obstacle sur le chemin qui empêche d'emprunter ce chemin). On retire alors cette tâche du plan. Cela contraint à retirer aussi deux liens causaux. On aboutit alors au plan suivant :

(b) Plan obtenu après le retrait d'une tâche de déplacement. Les instants fantômes et les liens fantômes sont représentés en vert.

Une fois tous les retraits effectués et avant de lancer une recherche (étape 5 du processus présenté sur la figure 5.2), on transforme ces liens fantômes en contraintes temporelles sur les éléments restant dans le plan. On aboutit alors au plan suivant :



(c) Plan obtenu après le retrait d'une tâche de déplacement. Les contraintes temporelles sont représentées en bleu.

Dans ce plan, il n'y a pas de nouvelle menace à détecter et le plan produit sera garanti de réaliser les actions provenant du plan initial dans l'ordre prévu initialement. Sans cela, l'action `move c d` n'aurait pas été contrainte après `move a b`.

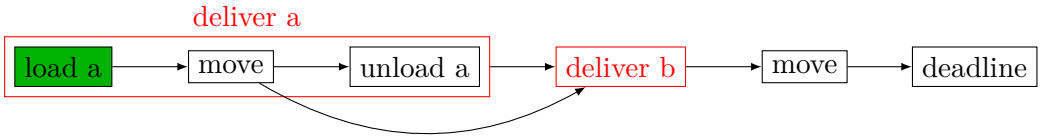
5.2.2 Utilisation de la hiérarchie des actions lors de la réparation

Le processus de réparation présenté jusqu'ici ne prend pas en compte la hiérarchie des actions et l'intention de l'utilisateur quand il a décrit les actions hiérarchiques. De plus, certains problèmes lors de l'utilisation des actions hiérarchiques peuvent se poser.

Exemple : Echec de la réparation à cause des actions hiérarchiques

Considérons le plan suivant, sur un problème de *logistics* :

Figure 5.5 – Exemple d'une réparation échouée à cause des actions hiérarchiques.

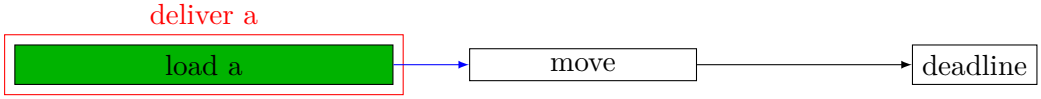


(a) Les tâches exécutées sont en vert et les actions hiérarchiques sont indiquées en rouge. Les flèches indiquent des liens causaux. La dernière tâche du plan représente une échéance : le véhicule doit être à une position donnée à une date précise. Les tâches de l'action hiérarchique `deliver b` n'ont pas été représentée pour ne pas surcharger la figure.

Un véhicule a donc prévu de faire deux livraisons avant de se rendre à un lieu donné pour respecter une échéance. Supposons maintenant que la tâche `load a` ait pris beaucoup de retard et que le véhicule n'ait plus le temps de faire ses deux livraisons avant l'échéance. Une réparation est alors nécessaire. On commence par retirer toutes les actions avant l'échéance :

(b) Plan obtenu après l'étape 4 de la réparation : on a retiré toutes les tâches précédant une échéance qui ne pouvait pas être respectée.

On se retrouve alors avec une action hiérarchique qui ne respecte plus sa méthode : on a exécuté l'action de chargement donc il n'est pas possible de l'enlever mais si on garde toutes les tâches on est pas sûr de pouvoir respecter l'échéance.
Pour respecter l'échéance, le planificateur va alors ajouter une action de déplacement :



(c) Plan partiel issu de la réparation. La flèche bleue représente une contrainte temporelle de précédence.

On arrive à un plan qui permet de respecter l'échéance mais le planficateur va avoir un problème : il n'a pas le droit d'ajouter d'action de déchargement sans qu'elle ne fasse partie d'une action hiérarchique. Cette action de chargement seule dans le plan empêche la description hiérarchique de faire son effet : la présence d'une partie

seulement d'une méthode crée un état qui n'a pas été prévu lors de la description des méthodes. Les méthodes disponibles ne permettent donc pas forcément de résoudre ce problème : si le planificateur n'a aucun moyen de décharger le véhicule il peut être impossible de résoudre le problème.

Pour respecter au maximum l'intention derrière la description des actions hiérarchiques et éviter les problèmes de complétude évoqués dans l'exemple précédent, il est nécessaire de limiter au maximum les méthodes partiellement instanciées. De plus, toutes les tâches d'une méthode ont été introduites en même temps pour accomplir un but donné et si une des tâches n'est plus disponible l'utilisation de cette méthode n'est sans doute plus pertinente.

Nous avons donc décidé d'utiliser les actions hiérarchiques présentes dans le plan lors du calcul du plan initial de la recherche. Pour chaque tâche qui doit être retirée du plan et qui fait partie d'une action abstraite, on désinstancie son action mère. Cela signifie que l'on retire tous les éléments de la méthode qui ont été introduits dans le plan (et un défaut abstrait est introduit). Si une tâche abstraite doit être retirée, alors on la désinstancie avant (*i.e.* on retire aussi tous ses enfants du plan).

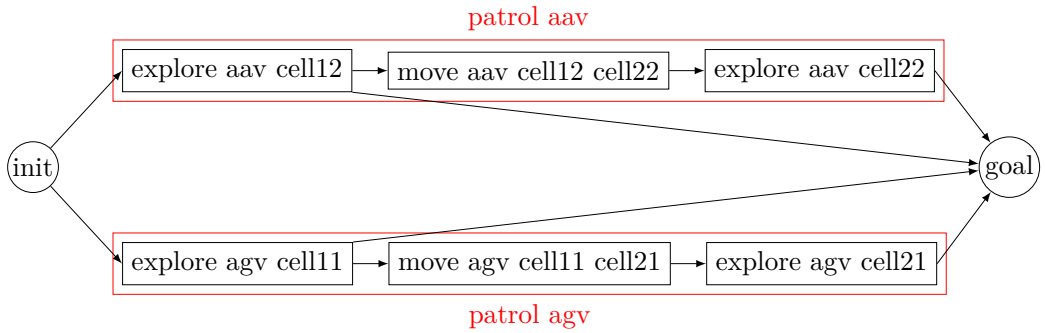
Si on doit retirer une tâche qui fait partie d'une méthode en cours d'exécution (*i.e.* dont au moins une tâche a été exécutée), on ne peut pas désinstancier complètement cette méthode. On retire alors uniquement cette tâche-là et on autorise le planificateur à ajouter n'importe quelle action dans le plan, même si la description hiérarchique l'interdit normalement. Cela permet au planificateur de trouver comment finir la méthode au mieux, au prix de l'augmentation de l'espace de recherche.

Si une méthode doit être désinstanciée (par exemple parce que l'on essaye de retirer son action abstraite) et qu'elle contient au moins une action imposée, alors il n'est pas possible de la désinstancier complètement. On procède alors de même : on retire tous les éléments possibles du plan et on autorise l'ajout de toutes les actions dans le plan.

Exemple : Retrait itératif d'éléments d'un plan hiérarchique

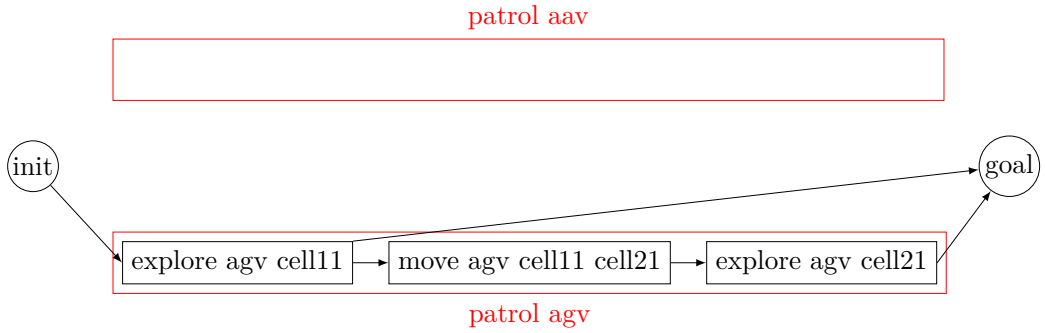
Considérons le plan suivant (repris de l'exemple de la page 131 avec des actions hiérarchiques) :

Figure 5.6 – Exemple de retrait itératif d'éléments d'un plan hiérarchique.



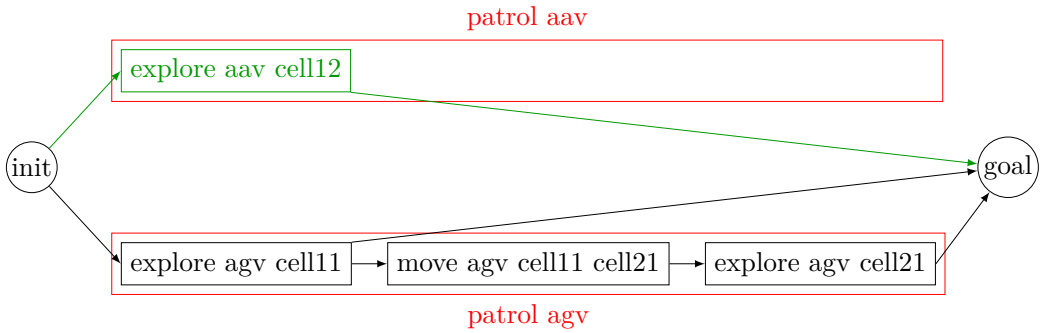
(a) Représentation du plan initial.

Supposons maintenant que l'action `move aav cell12 cell22` soit indisponible. Elle est retirée du plan et comme elle fait partie d'une action abstraite, toute l'action abstraite est désinstanciée. Le plan initial pour la recherche est alors :



(b) Plan initial de la recherche si `move aav cell12 cell22` est indisponible.

L'action abstraite est toujours présente mais une autre méthode devra être choisie pour la remplacer.
Si la première action de l'action hiérarchique `patrol aav` était déjà exécutée, il ne serait pas possible de désinstancier l'action abstraite. On retirerait alors tous les éléments possibles et les buts de l'action seraient accomplis en introduisant de nouvelles actions élémentaires.
On aboutirait au plan suivant :



(c) Plan initial de la recherche si `move aav cell12 cell22` est indisponible et que l'exécution de l'action abstraite a déjà commencé. Les actions indiquées en vert sont exécutées (ou en cours d'exécution).

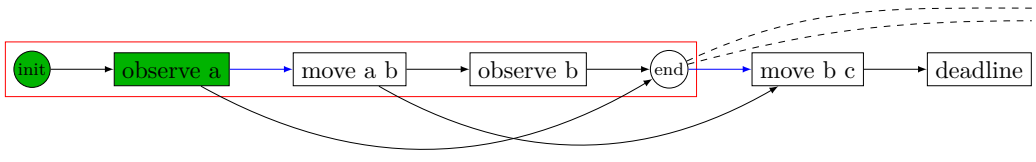
Contraction des tâches finales. Un problème peut encore se poser lorsque l'on retire certains éléments d'une méthode. Chaque méthode, en tant que plan partiel, contient une tâche finale. Cette tâche particulière utilise les effets de l'action abstraite comme ensemble de préconditions et d'effets. Lors du retrait de toutes les tâches non exécutées d'une méthode, on peut alors se poser la question du retrait de cette tâche particulière.

Nous avons décidé de garder dans le plan cette tâche afin de conserver la cohérence de la méthode en tant que plan partiel. Néanmoins cela introduit de nouveaux liens ouverts à établir avant l'échéance et cela peut empêcher la réparation comme le montre l'exemple suivant.

Exemple : Nécessité de la contraction de la tâche finale d'une méthode

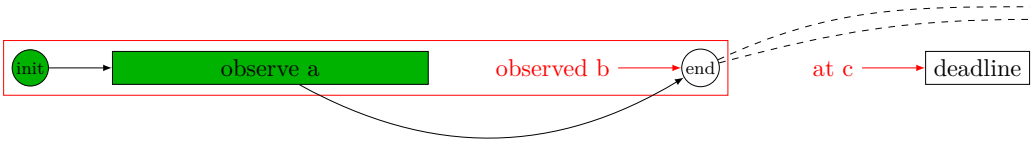
La figure 5.7a présente un extrait d'un plan en cours d'exécution. On se concentre sur les tâches précédant une échéance, les actions suivant cette échéance ne sont pas représentées.

Figure 5.7 – Exemple d'une réparation nécessitant la contraction d'une tâche finale.



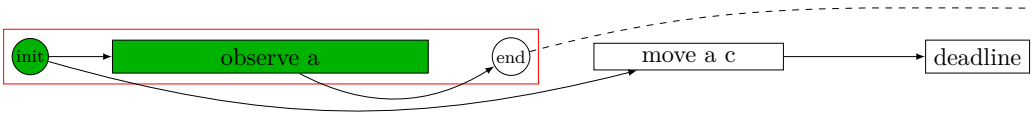
(a) Plan en cours d'exécution. La tâche d'observation de `a` est en cours d'exécution. Le robot doit être en `c` à une date donnée, ce qui est imposé par la présence de la tâche dénommée `deadline`. Le plan prévoit d'aller explorer `b` avant d'aller en `c`. L'action hiérarchique utilisée explore `a` et `b`. Les liens causaux sont représentés par des flèches noires et les contraintes de précédences temporelles par des flèches bleues. Les traits en pointillé représentent des liens causaux vers la tâche de fin globale du plan : ce sont des buts à atteindre.

Supposons que lors de l'exécution de l'action `observe a`, le robot prend plus de temps que prévu. Cela peut être dû à un problème au niveau des capteurs qui a nécessité de refaire des acquisitions par exemple. Lorsque ce retard est observé, les contraintes temporelles du plan ne sont plus respectées : il n'est plus possible d'aller observer `b` avant de se rendre en `c` pour tenir l'échéance.
On tente donc de réparer le plan. Comme une échéance n'est pas respectée, on retire toutes les tâches possibles du plan situées avant l'échéance. Pour garder la méthode cohérente en tant que plan partiel, on garde sa tâche finale. On aboutit alors au plan suivant :



(b) Plan partiel obtenu par retrait de toutes les tâches non exécutées présentes avant l'échéance. Les liens ouverts sont représentés en rouge.

La tâche finale que l'on a laissée a produit un lien ouvert car une de ses préconditions n'est plus garantie. De plus, les contraintes temporelles du plan initial font que cette tâche doit être effectuée avant l'échéance. Or aucun plan ne peut réaliser cet objectif. Il est donc nécessaire d'abandonner cet effet de la méthode : on le retire des préconditions et des effets de la tâche finale de la méthode. Cela revient à retirer le lien causal qui garantit l'exploration de `b` et le lien ouvert vers cette tâche.
On aboutit alors à un plan qui peut être réparé comme suit :



(c) Plan réparé. Cela a nécessité l'abandon d'une précondition de la tâche finale de la méthode utilisée.

Les actions antérieures, en particulier celles qui garantissent que `b` est bien exploré, ne sont pas représentées ici. Mais elles doivent être présentes pour que le plan soit valide.

Pour garantir la réparation, il est donc nécessaire de retirer des tâches finales les effets qui ne sont pas déjà garantis par des liens causaux dans le plan. Ce processus est appelé *contraction*. Il permet de ne pas imposer aux méthodes des plans en cours de réparation de satisfaire tous les effets attendus d'elles pour ne pas sur-contraindre le problème.

Définition explicite des fluents de bas niveau. Dans l'exemple qui a été présenté jusqu'ici, l'idée guidant l'utilisation des actions hiérarchiques a été de permettre un raisonnement à haut niveau (sur des équipes de robots, des zones contenant plusieurs cellules, etc.) pour produire un plan abstrait. Un fois obtenu, ce plan peut être raffiné en

un plan solution, après l'avoir complété pour assurer les contraintes concernant les robots individuels et les cellules individuelles.

Jusqu'ici, les différences entre fluents de haut niveau et fluents de bas niveau n'ont pas été explicitement décrites : le planificateur raisonne sur les buts et les préconditions des actions qu'il ajoute au plan. Un fluent qui n'apparaît dans la recherche qu'après l'obtention d'un plan abstrait (comme les positions des robots individuels dans l'exemple de la page 81) est donc implicitement un fluent de bas niveau. Pour assurer que la recherche soit possible, il a été nécessaire de résoudre les actions abstraites dans l'ordre chronologique du plan (cf. page 81). Cela empêche un lien ouvert d'un fluent de bas niveau de se trouver après, dans le plan, une action hiérarchique non instanciée. Néanmoins, cela n'est plus suffisant dans le cas de la réparation.

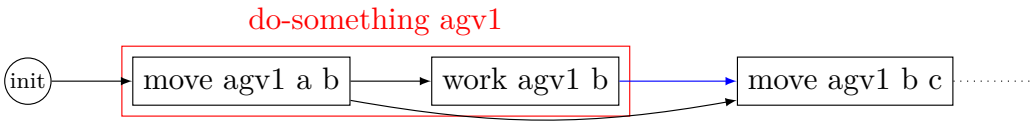
En effet, lors du retrait de certaines tâches du plan (devenues obsolètes par exemple), tous les liens causaux les impliquant sont retirés. On peut donc se retrouver avec des liens ouverts de bas niveau. Il suffit d'ajouter une action abstraite à la place des tâches retirées pour se retrouver dans le cas où une action abstraite non instanciée précède un lieu ouvert qui doit être résolu après son instanciation.

C'est pourquoi nous avons choisi de permettre à l'utilisateur de spécifier une famille de fluents dont les liens ouverts doivent être résolus après l'instanciation des actions abstraites.

Exemple : Fluents de bas niveau lors d'une réparation

Dans un contexte de réparation, une fois certaines tâches retirées, il est possible d'insérer des tâches à l'endroit laissé vide dans le plan. On peut alors se trouver avec la situation décrite ci-dessous.
Supposons le plan initial suivant :

Figure 5.8 – Exemple de plan nécessitant l'utilisation des fluents de bas niveau.

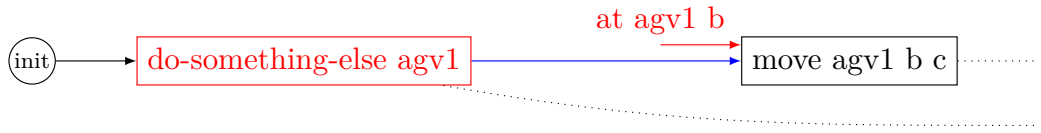


(a) Plan initial. Les liens causaux sont indiqués en noir et les liens temporels en bleu. Par souci de clarté, le reste du plan n'a pas été représenté.

Ce plan commence donc par une action abstraite. Le choix de la méthode d'instanciation détermine la position finale du robot (comme dans les exemples de patrouilles qui ont été présentés dans l'exemple de la page 81). Une fois les actions abstraites instanciées, les actions de déplacement individuel des robots sont ajoutées. Le calcul du plan abstrait n'a donc pas besoin de raisonner sur les positions des robots.
Si l'action `do-something agv1` n'est plus disponible, alors elle est retirée du plan. Cela crée un lien ouvert pour l'action `move agv1 b c`. On aboutit alors au plan suivant :

(b) Plan partiel utilisé comme point de départ de la recherche si l'action `do-something agv1` n'est plus disponible. Les liens ouverts sont représentés en rouge.

Pour accomplir les objectifs du plan, le planificateur peut être amené à introduire une nouvelle action abstraite. Cette action abstraite ayant besoin de déplacer le robot, elle est en conflit avec tous les fluents de position du robot. On détecte donc une menace avec l'action `move agv1 b c` déjà présente dans le plan. Supposons que le planificateur décide de réaliser cette nouvelle action abstraite avant `move agv1 b c`. On aboutit alors au plan suivant :



(c) Plan partiel en cours de réparation.

Une fois dans cette situation, si le planificateur décide de résoudre le lien ouvert, alors il ne peut pas trouver de solution. Il lui est impossible d'établir un lien causal avec une action (même une nouvelle action introduite pour l'occasion) précédant l'action hiérarchique non instanciée à cause des conflits de cette action. L'action hiérarchique elle-même ne produit pas de fluent concernant la position des robots, c'est le choix de la méthode qui déterminera la position finale des robots. Il faut donc attendre l'instanciation de l'action hiérarchique pour voir apparaître les actions de déplacement des robots individuels. Et ajouter une action de déplacement après l'action abstraite ne changera pas le problème : elle introduira elle aussi un lien ouvert de position et on se retrouvera dans le même cas que précédemment.
Il est donc nécessaire d'instancier l'action abstraite (donc de résoudre le défaut abstrait) avant de résoudre le lien ouvert sur la position du robot. Dans ce cas on peut définir la famille des fluents de position des robots comme des fluents de bas niveau : leurs liens ouverts seront alors résolus après les défauts abstraits.

En résumé, l'ordre de résolution des défauts est le suivant :

- les menaces ;
- les liens ouverts (autres que ceux de bas niveau) ;
  - Si on utilise *local*, on commence par ceux introduits par l'action introduite le plus récemment. Si on utilise *sorted*, on commence par les plus coûteux au sens de h^{add}.
  - Si on utilise *earliest*, les égalités sont départagées en choisissant en priorité les plus tôt dans le plan.

  - Si on utilise *motionLast*, les liens ouverts de position sont résolus après tous les autres.
- les défauts abstraits (triés par date dans le plan si l'heuristique *earliest* est utilisée) ;
- les liens ouverts de bas niveau.

5.3 Adaptation pour la réparation distribuée sans garantie de communication et en cours d'exécution

Les techniques présentées jusqu'ici permettent de réparer un plan en cours d'exécution. Elles garantissent que les actions déjà exécutées sont bien toutes présentes dans le plan solution, que toutes les actions non exécutées sont dans le futur et que le plan produit est bien une solution du problème. Néanmoins, cela ne suffit pas pour notre application.

5.3.1 Définition des agents d'une action

En effet, les communications entre tous les robots ne sont pas forcément garanties. Cela implique qu'une réparation peut avoir lieu alors que certains robots sont hors de portée de communication. Dans un tel cas, il n'est pas possible d'ajouter ou d'enlever des actions à ces robots hors de portée : même si le plan prévoit des changements aux actions de ces robots il ne pourra pas être exécuté tant que la communication n'est pas rétablie. Il est donc nécessaire de pouvoir garantir que le plan produit contienne toutes les actions (passées et futures) des robots avec lesquels la communication n'est pas possible.

Nous avons donc ajouté dans la description PDDL des actions (cf. section 1.1.3) le nom de l'agent responsable de l'action.

Exemple : Syntaxe d'une action appartenant à un agent

Une action de déplacement peut alors être décrite comme suit :

```
1 (:durative-action move
2   :parameters (?r - robot ?from ?to - loc)
3   :agent (?r)
4   :duration (= ?duration (distance-aav ?from ?to))
5   :condition (and (at start (at ?r ?from))
6                   (over all (adjacent ?from ?to)))
7   :effect (and (at end (at ?r ?to))
8                (at start (not (at ?r ?from))) )
9 )
```

La ligne 3 définit l'appartenance de cette action au robot devant la réaliser.

Cette nouvelle information est facultative, mais lors d'une réparation le planificateur peut recevoir la liste des agents impliqués dans la réparation. Toutes les actions n'appartenant pas à un agent de cette liste sont bloquées dans le plan : elles ne peuvent pas être retirées lors du calcul du plan de départ de la recherche présenté à la figure 5.2. Les actions bloquées sont donc considérées de la même manière que les actions exécutées. De plus, lors de l'ajout d'une action pour résoudre un lien ouvert, les actions appartenant à un agent qui n'est pas impliqué dans la réparation sont interdites. Cette modification revient

à une modification du domaine (au sens de PDDL) car elle modifie l'ensemble des actions disponibles pour le planificateur.

Cet ajout minimaliste permet de satisfaire nos contraintes. D'autres formalismes plus poussés ont néanmoins été proposés [Kovacs 2012] qui intègrent plus d'information : partitionnement des objets entre les différents agents, définition de buts associés à chaque agent, etc. Nous avons décidé de rester à un formalisme plus simple qui permette de satisfaire ces contraintes sans modifier en profondeur notre algorithme.

5.3.2 Relâchement des buts de haut niveau

Une autre modification concerne le critère d'admissibilité des plans. Jusqu'à présent, un plan n'est considéré comme valide que s'il satisfait tous les buts du problème (cf. définition 13 page 19). Néanmoins, dans le cadre d'une mission complexe pendant laquelle des robots peuvent tomber en panne, il peut devenir impossible de réaliser certains buts. Si un objectif ne peut être réalisé que par un robot et que ce robot tombe en panne, aucun plan ne parviendra à satisfaire la mission, même si tous les autres objectifs sont atteignables.

Nous avons donc ajouté une option à HiPOP qui permet d'ignorer certains buts inatteignables du problème. Si cette option est passée à HiPOP, tout lien ouvert vers la tâche finale du plan qui n'a aucun résolveur est ignoré. Ainsi, le plan reste cohérent (toutes les préconditions de toutes les actions sont garanties) mais tous les buts peuvent ne pas être atteints. Cela permet de rapidement rejeter les buts inatteignables sans compromettre l'intégrité du plan. Néanmoins, le lien ouvert sera conservé si une action accomplissant ce but existe, même si aucun plan contenant cette action n'est faisable (à cause d'une échéance notamment).

Cette option n'est pas utilisée dans les benchmarks qui suivent, elle ne sera utilisée que lors de la réparation d'un plan pendant l'exécution d'une mission. Elle permet de ne pas abandonner la mission si un de ses objectifs n'est plus possible.

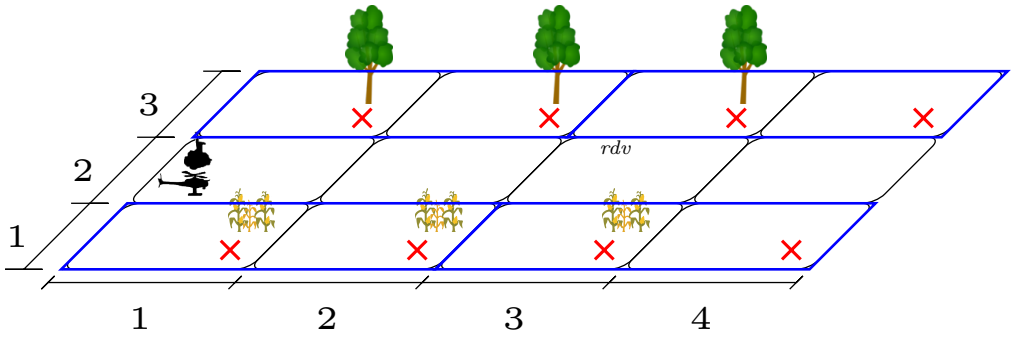
5.4 Exemples de réparation

L'algorithme de réparation présenté est illustré par deux exemples : une réparation suite à un retard empêchant une échéance d'être respectée et une réparation suite à la casse d'un robot.

Exemple : Exemple de réparation suite à un retard

Considérons le problème suivant :

Figure 5.9 – Exemple de réparation suite à un retard.



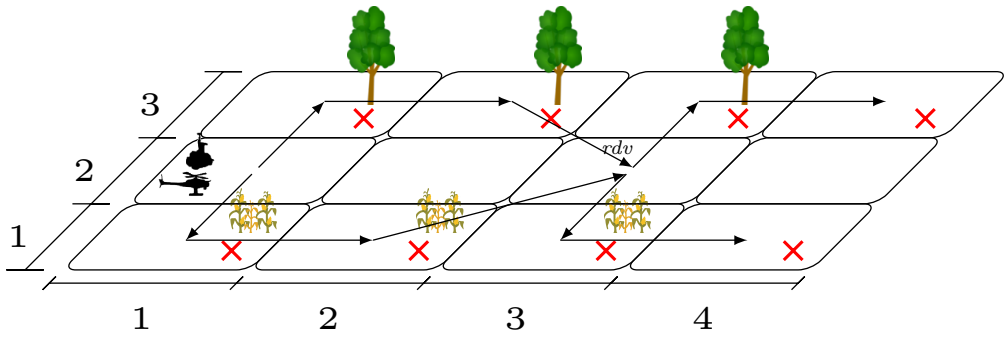
(a) État initial : deux robots sont disponibles, 8 cellules sont à explorer et un rendez-vous est prévu à la cellule indiquée sur la figure. Chaque cellule est désignée par sa coordonnée en x puis en y. Ainsi, la cellule de départ est `cell12`.

Le but est d'observer les 8 cellules indiquées par une croix rouge et d'assurer un rendez-vous dans la cellule `cell32`. Toutes les actions de déplacement entre cellules adjacentes et les actions d'observation ont une durée de 1. Les actions de déplacement entre cellules non-adjacentes sont possibles et utilisent la distance de Manhathan : elles sont ainsi équivalentes au chemin utilisant les actions de déplacement entre cellules adjacentes. Les cellules à observer sont regroupées en 4 zones de 2 cellules chacune (`cell11` et `cell21`, `cell31` et `cell41`, `cell13` et `cell23`, `cell33` `cell43`). Deux zones sont uniquement composées de forêt ou de champ : il y a donc un seul véhicule qui peut les explorer. Les deux autres ont une cellule explorable par tous les robots et une cellule réservée. On fournit au planificateur 10 patrouilles, 5 pour chacun des robots. 4 patrouilles correspondent à l'exploration des deux zones auxquelles il a complètement accès, dans les deux sens d'exploration possibles. La dernière correspond à l'exploration de la cellule unique à laquelle le robot a accès dans l'autre zone. Ainsi le drone aérien possède les 5 patrouilles suivantes :

- `cell11` puis `cell21`
- `cell21` puis `cell11`
- `cell31` puis `cell41`
- `cell41` puis `cell31`
- `cell43`

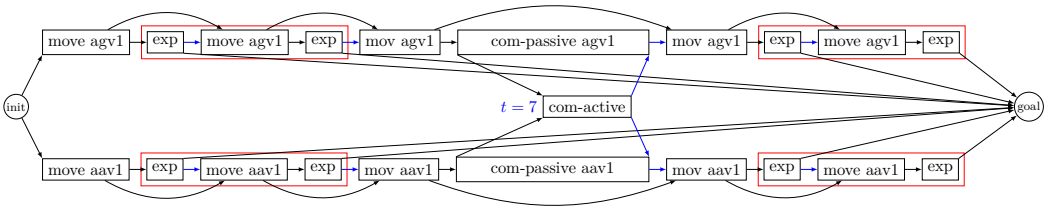
Le rendez-vous est imposé à $t = 7$ pour permettre la réalisation d'une une patrouille avant et imposer au moins une patrouille après. Pour séparer les actions des différents agents, le rendez-vous est représenté par deux actions élémentaires distinctes. La première, `com-passive`, est assignée à un robot et représente le fait qu'il est à la position voulue, prêt à communiquer. Elle crée un fluent au début de l'action qui signale que le robot est prêt et le détruit à la fin de l'action. La deuxième, `com-active`, nécessite d'être concurrente avec l'action `com-passive` de chaque robot. C'est elle

qui a une échéance fixe et qui peut être assignée à un agent virtuel représentant l'environnement ou à un des robots indifféremment. Le planificateur peut introduire des `com-passive` pour satisfaire une action `com-active` déjà dans le plan, mais il ne va pas en ajouter une.
La description en PDDL de cet exemple est fournie à l'annexe D.
La trajectoire des robots dans le plan initial calculé par HiPOP est représentée sur la figure 5.9b.



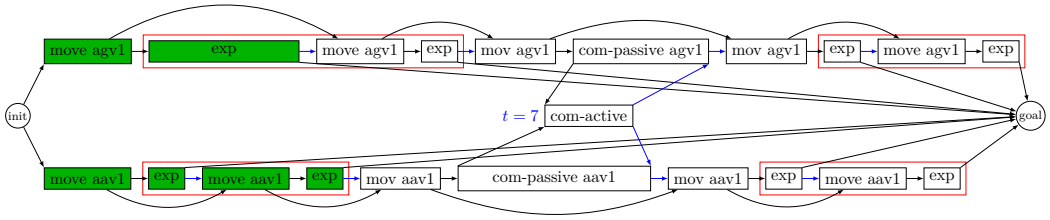
(b) Plan initial.

La figure 5.9c présente une représentation du plan initial comprenant toutes les tâches, liens causaux, liens temporels et actions hiérarchiques. Pour limiter la taille de la figure, les noms complets des actions ont été remplacés par des abréviations.



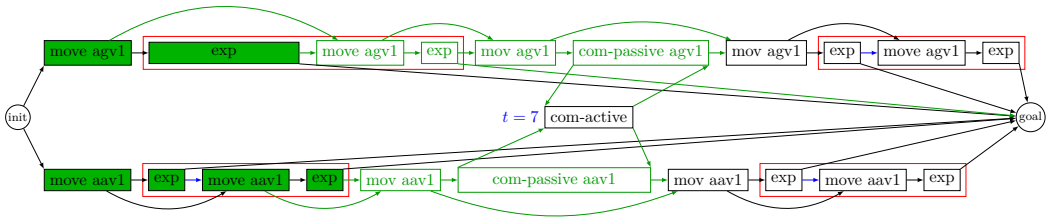
(c) Plan initial. Les tâches sont représentées par des rectangles (sauf les deux tâches particulières de début et de fin). Les liens causaux sont en noir et les liens temporels en bleu. Chaque robot est associé à une ligne, les actions communes étant au milieu.

On suppose maintenant que pendant l'exécution de ce plan, un retard survient. Plus précisément, lors de la première exploration de l'AGV, le robot prend plus de temps que prévu : 2 unités de temps au lieu d'une seule. Pendant ce temps, l'AAV a eu le temps de finir sa première observation, de faire son deuxième déplacement et vient juste de commencer sa deuxième exploration. On aboutit alors au plan représenté ci dessous :



(d) Plan initial en cours d'exécution. Les actions exécutées (ou en cours d'exécution) sont représentées sur fond vert.

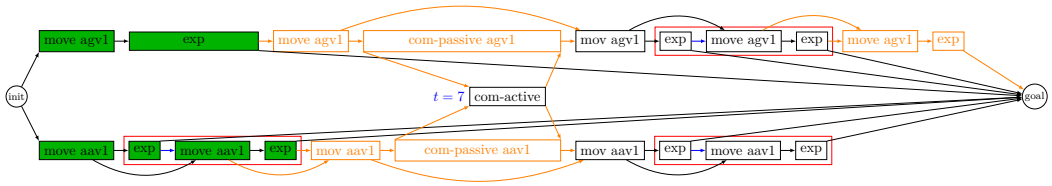
On voit que ce retard a décalé temporellement toutes les actions de l'AGV à tel point que la précédence de l'action `com-passive` sur l'action `com-active` n'est plus respectée. Concrètement, cela signifie que l'AGV n'a plus le temps d'exécuter toutes les actions prévues pour être à l'heure à son rendez-vous avec l'AAV. Il est donc nécessaire de réparer le plan.
La première étape de la réparation consiste à retirer des tâches pour calculer le plan de départ de la recherche. Ici, il n'y a pas de tâches obsolètes mais une échéance n'est pas respectée. On retire donc toutes les tâches entre la date actuelle et l'échéance. Il est bien entendu interdit de retirer l'action dont la présence est imposée dans le plan et qui porte l'échéance. Tous les tâches retirées sont indiquées en vert sur la figure 5.9e



(e) Plan en cours de réparation. Les éléments retirés du plan lors de la réparation sont indiqués en vert.

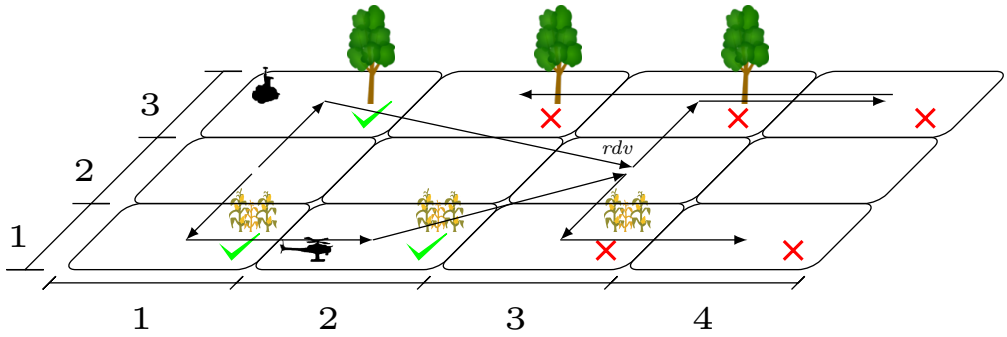
Comme on peut le voir, aucune distinction n'est réalisée entre les tâches des deux robots. Des liens ouverts sont introduits pour chaque lien causal retiré qui pointait vers une tâche non retirée. Les liens fantômes (cf. page 132) assurent que toutes les tâches non retirées sont bien prévues après la tâche `com-active` car elles l'étaient dans le plan initial.
Une fois toutes les tâches retirées, la recherche peut commencer et on fait appel à HiPOP. Les tâches retirées à l'AAV peuvent être directement ré-introduites dans le plan. Du coté de l'AGV, on peut ajouter la tâche de communication `com-passive` et la tâche de déplacement nécessaire pour aller directement de la position initiale de l'AGV au lieu de rendez-vous. Pour satisfaire les buts, il est néanmoins nécessaire d'observer la cellule `cell23`. Comme on a retiré des tâches à une action hiérarchique sans pouvoir retirer toutes les tâches filles de cette action, le planificateur a le droit d'ajouter n'importe quelle action élémentaire. Le planificateur a donc ajouté une

action d'exploration seule (*i.e.* en dehors de toute action hiérarchique). On aboutit alors au plan suivant :



(f) Plan réparé. Les nouveaux éléments sont indiqués en orange.

La trajectoire des robots est alors :



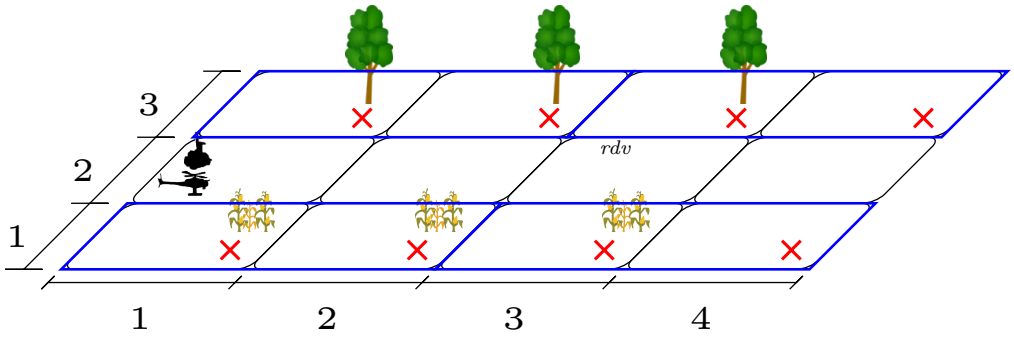
(g) Plan réparé après le retard d'une action.

La nouvelle patrouille aurait tout aussi bien pu être introduite avant la patrouille déjà présente dans le plan et après la communication.
Sur la figure 5.9f, tous les éléments en noir sont les éléments qui sont explicitement gardés du plan initial dans le plan réparé. Dans un plan plus complexe, cela peut représenter bien plus d'actions mais la recherche restera la même : si le plan est plus grand, la recherche n'augmente pas en complexité.

Exemple : Exemple de réparation suite à un robot inopérant

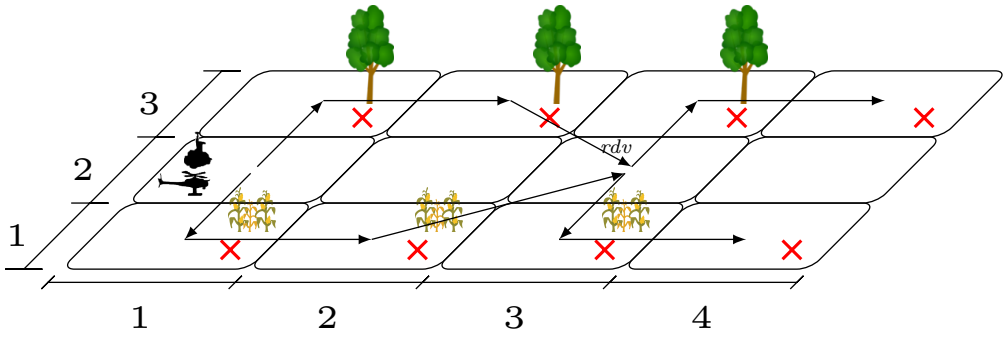
Considérons le problème suivant (identique à l'exemple précédent) :

Figure 5.10 – Exemple de réparation suite à un robot inopérant.

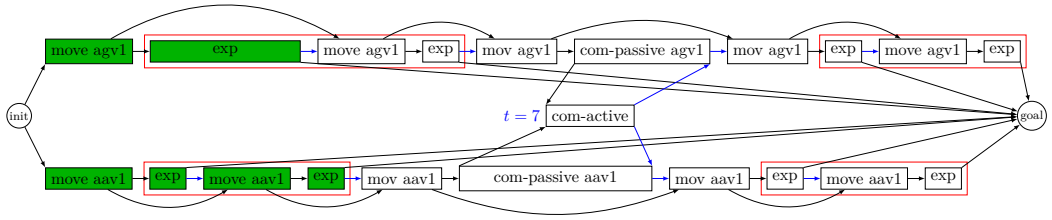


(a) État initial : deux robots sont disponibles, 8 cellules sont à explorer et un rendez-vous est prévu à la cellule indiquée sur la figure. Chaque cellule est désignée par sa coordonnée en x puis en y. Ainsi, la cellule de départ est `cell12`.

On se place dans le même cas que dans l'exemple précédent : on a commencé l'exécution du plan et un problème survient lors de la première exploration de l'AGV. Au lieu de juste retarder l'action, on considère maintenant que le robot est devenu inopérant. Au bout de 2 unités de temps, il détecte un problème qu'il ne peut pas surmonter et en informe l'autre robot. L'AAV doit donc continuer seul la mission. Le plan initial est le même que dans l'exemple précédent. Les trajectoires sont représentées à la figure 5.10b et les actions à la figure 5.10c.

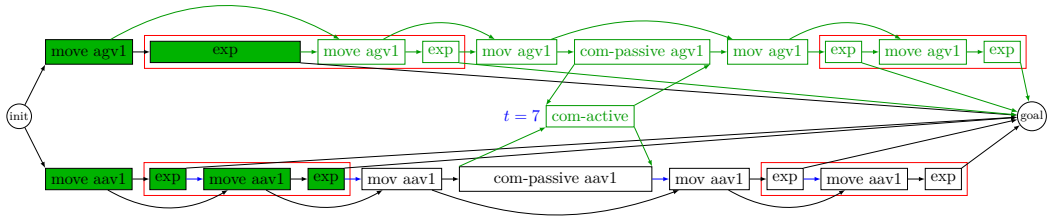


(b) Plan initial.



(c) Plan initial en cours d'exécution. Les actions exécutées (ou en cours d'exécution) sont représentées sur fond vert.

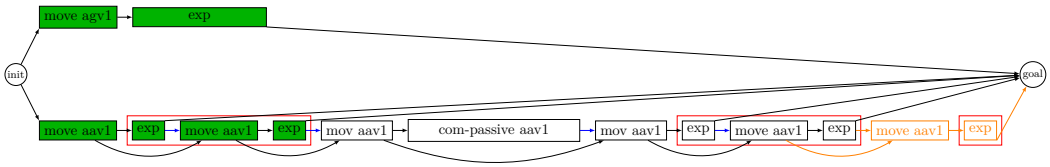
Lors de la création du plan de départ de la recherche, la première étape consiste à retirer toutes les actions obsolètes. Ici, toutes les actions non-exécutés de l'AGV sont obsolètes. De plus, tous les buts de la mission ne sont plus réalisables : le rendez-vous n'est plus possible et tous les points ne sont pas explorables.
Concernant les points à explorer, il n'est pas nécessaire de savoir lesquels sont explorables ou non. Le réglage décrit dans la section 5.3.2 permet de signaler au planificateur que l'on souhaite réaliser le plus de buts possible sans tous les imposer. On décide alors d'utiliser cette option.
En ce qui concerne le rendez-vous, l'action `com-active` est d'office obligatoire dans le plan car elle représente un but du problème. Ici, aucun plan possédant cette action ne peut être valide car l'AGV ne peut pas réaliser sa part du travail. Il est donc nécessaire de retirer l'action `com-active` du plan. Cela n'est pas réalisable automatiquement par HiPOP car cela ne concerne pas la suppression d'un lien ouvert vers la tâche de fin d'un plan qui n'a pas de résolveur possible. Il faut donc retirer explicitement la tâche du plan pour signaler à HiPOP de laisser tomber cet objectif.
La première étape de la réparation va donc consister à retirer tous les éléments verts présenté sur la figure 5.10d.



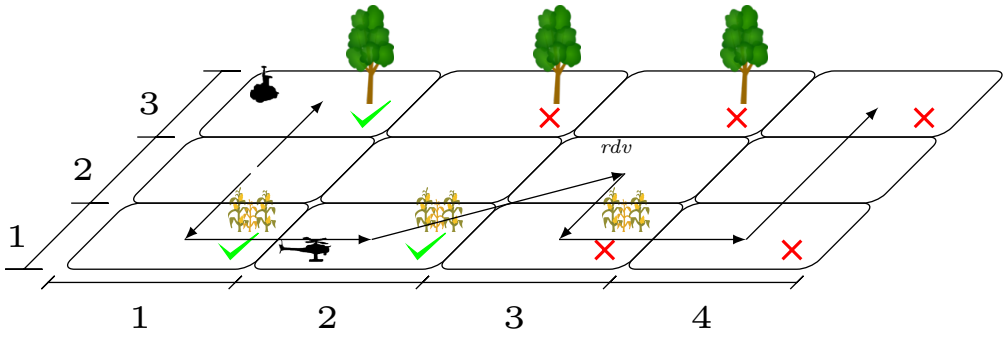
(d) Plan en cours de réparation. Les éléments retirés du plan lors de la réparation sont indiqués en vert.

Comme on peut le voir ici, on ne retire pas du plan la tâche `com-passive` de l'AAV ni les tâches de déplacement qui permettent de se rendre au point de rendez-vous. Cela veut dire que le plan produit sera sous-optimal : il contiendra des tâches inutiles restant du plan initial. Ce sont des conséquences des choix qui ont été faits pour trouver un plan le plus proche possible du plan initial. En revanche, si l'algorithme de réparation venait à itérer une fois (cf. figure 5.2), alors on retirerait la tâche

`com-passive` de l'AAV du plan car elle est liée causalement à une action retirée. Une deuxième itération retirerait les tâches de déplacement associées et permettrait de trouver un meilleur plan.
Après le retrait des tâches obsolètes du plan, on peut lancer une recherche. Le seul lien ouvert qui peut être résolu par l'AAV et qui ne l'est pas déjà est l'exploration de la cellule `cell43`. Pour ce faire, le planificateur ajoute l'action abstraite correspondante et l'instancie (la méthode étant constituée d'une unique tâche).
On aboutit alors au plan réparé suivant (dont les trajectoires sont représentées à la figure 5.10f) :



(e) Plan réparé. Les nouveaux éléments sont indiqués en orange.



(f) Plan réparé après la casse d'un robot.

5.5 Validation expérimentale de la réparation

Intéressons-nous maintenant à la validation expérimentale de cet algorithme de réparation. Dans un premier temps, nous avons voulu comparer la réparation et la replanification. Pour cela, nous avons utilisé un benchmark adapté tiré du domaine *survivors* auquel nous avons déjà eu recours pour les benchmarks précédents. Nous avons donc créé un ensemble de problèmes initiaux, de plans initiaux et de problèmes à résoudre Ces problèmes à résoudre sont construits en modifiant les problèmes initiaux. Les problèmes consistent donc à réparer un plan directement produit par HiPOP, et la comparaison avec d'autres planificateurs est possible car la réparation est alors uniquement vue comme un problème de planification. La réparation est effectué sur un plan qui n'est pas en cours d'exécution (*i.e.* aucune action exécutée ou en cours d'exécution n'est imposée dans le plan) et aucune échéance n'est présente dans les buts.

Pour valider complètement notre algorithme de réparation, il est aussi nécessaire de le faire fonctionner dans des cas de réparation avec un plan en cours d'exécution. Pour cela, il faut un simulateur d'exécution pour produire les plans à réparer. Nous n'aborderons donc pas ces tests ici mais dans le prochain chapitre, en même temps que l'algorithme d'exécution.

5.5.1 Benchmark survivors

Pour ce benchmark, nous avons généré en même temps le problème initial et le problème. Le problème est obtenu en modifiant le problème initial suivant un aléa. Nous avons défini trois types d'aléa :

- Un nouveau blessé apparaît (*newSurvivor*) : un nouveau but apparaît dans le domaine
- Un blessé change de position (*survivorPos*) : un des buts actuels du plan est retiré et remplacé par un autre
- La position initiale des robots change (*startingPos*) : les fluents disponibles dans l'état initial changent

Une fois les problèmes initiaux générés, HiPOP est appelé pour générer un plan initial. Les tests portent alors sur la réparation : on fournit à HiPOP le problème et le plan initial. Les configurations utilisant le plan initial pour réparer sont appelées *repair*.

On cherche à répondre à deux questions :

- Quel est l'intérêt de la réparation vis-à-vis de la replanification ?
- Quelle est l'influence de l'utilisation des actions hiérarchiques sur la réparation ?

Quel est l'intérêt de la réparation vis-à-vis de la replanification ? Comme observé dans les chapitres précédents, les configurations utilisant *areuse* et *reuse* présentent des résultats similaires. De même, *local* et *sorted* sont très proches. Dans ce qui suit, nous nous sommes donc concentrés sur les configurations *local* et *areuse*.

La figure 5.11 présente une comparaison entre les configurations effectuant une réparation (étiquetée *repair*) et celles effectuant une replanification.

Parmi les configurations utilisant *repair*, les versions utilisant *motion* ou *ncm* sont assez proches. Pour les configurations faisant de la replanification, on a les mêmes résultats que dans le chapitre 4 : les configurations utilisant *ncm* résolvent moins de problèmes que celles utilisant *motion*. Dans tous les cas de figure, les configurations effectuant une réparation sont plus efficaces que la replanification : elles résolvent plus rapidement les problèmes (les courbes des configurations *repair* sont au-dessus des autres sur la première ligne de la figure 5.11).

On observe aussi que la stabilité du plan est conservée : dans tous les cas de réparation (sauf 1) le nombre d'actions changées est inférieur à 20 (sur la dernière ligne de la figure 5.11). Lors de la replanification, le nombre de changements est fortement lié au nombre d'actions dans le plan : on n'a aucune garantie qu'une action du plan initial se retrouve dans le plan produit.

En terme de durée du plan produit (deuxième ligne de la figure 5.11), toutes ces configurations produisent des plans de durée équivalente. Même la réparation, qui pourrait voir la durée des plans produits négativement impactée à cause de la conservation des actions présentes dans le plan initial, ne fait pas augmenter la durée des plans produits.

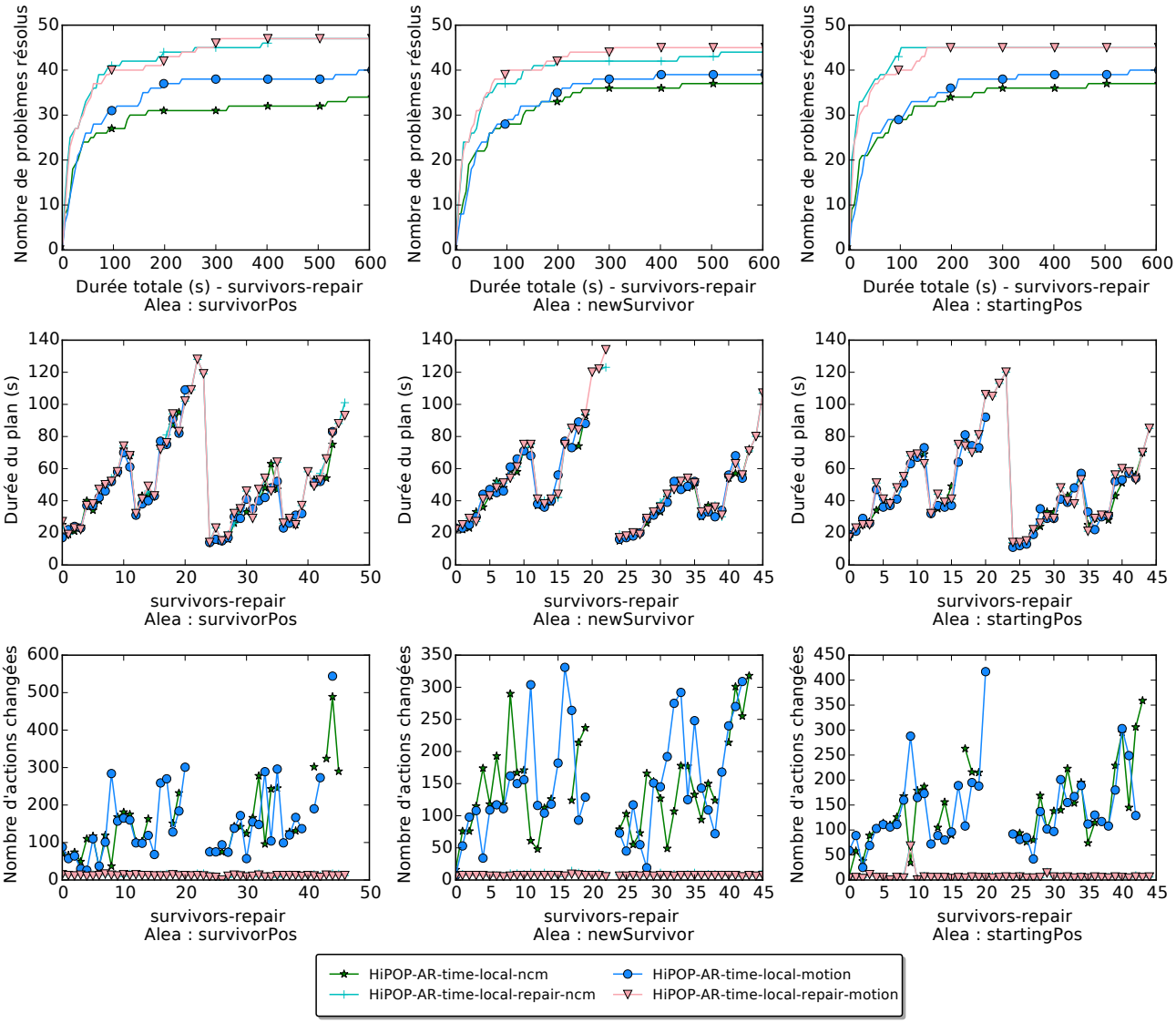


Figure 5.11 – Comparaison de plusieurs planificateurs sur le nombre de problèmes résolus, la durée du plan trouvé et la stabilité du plan. Chaque colonne correspond à un aléa différent. La première ligne présente le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse. La deuxième ligne présente la durée de chaque plan trouvé. La troisième ligne présente la stabilité de chaque plan : le nombre de tâches ayant été enlevées ou rajoutées entre le plan trouvé et le plan initial.

En résumé, la réparation présentée ici permet d'améliorer les performances de l'algorithme de planification sous-jacent. Comme attendu le plan trouvé est proche du plan initial, mais en plus il est trouvé plus rapidement et sa durée n'est pas augmentée.

Quelle est l'influence de l'utilisation des actions hiérarchiques sur la réparation ? Pour pouvoir étudier l'influence des actions hiérarchiques présentes dans le plan initial, nous avons aussi généré des plans initiaux sans utiliser la hiérarchie. Cela limite néanmoins le nombre de plans initiaux trouvés (comme on l'a vu au chapitre 4) ce qui

limite le nombre de tests possibles. Au vu du nombre de plans initiaux produits (28 sur les 144 problèmes initiaux), nous avons préféré utiliser une autre stratégie. Les mêmes plans initiaux sont utilisés mais la hiérarchie est ignorée : dès l'importation du plan on retire toutes les informations concernant les actions abstraites du plan. Le planificateur a encore accès aux actions hiérarchiques pour planifier mais elles ont été retirées du plan initial. Cette configuration est appelée *repairFlat*.

La figure 5.12 présente une comparaison entre les configurations utilisant *motion* effectuant une réparation, une réparation depuis un plan non abstrait et une replanification. Les configurations utilisant *ncm* pour la réparation sont sensiblement équivalentes.

On observe alors des différences selon les aléas introduits.

Quand l'état initial est le seul à changer (*startingPos*, colonne de droite), la réparation consiste à remplacer les premières actions de déplacement du plan. Dans ce cas la hiérarchie des actions n'apporte rien de particulier car seule la première action de chaque robot est impactée.

Dans le cas où on ajoute un but (*newSurvivor*, colonne du milieu), il est plus efficace de partir du plan initial contenant la hiérarchie que du plan complètement élémentaire. Dans certains cas, la réparation doit itérer une fois sur le plan de départ de la recherche car le premier plan testé n'a pas abouti. Cela fait perdre du temps à la recherche, et si plus d'itérations sont nécessaires il n'y a pas forcément le temps pour le faire.

Les résultats varient le plus dans le cas où un but devient obsolète (*survivorPos*, première colonne). La présence des actions hiérarchiques permet de retirer en une seule fois toutes les actions qui ont été introduites pour satisfaire chaque but, et quand ce but n'est plus d'actualité on peut toutes les retirer ensemble. Sans cette information, il est même plus rapide de replanifier complètement plutôt que de réparer le plan. Ces résultats s'expliquent par le fait que le planificateur échoue à trouver une solution lors du premier essai et qu'il est nécessaire de faire un deuxième essai pour trouver une solution. L'absence des actions hiérarchiques fait que l'estimation des actions à retirer du plan n'est pas assez complète.

De plus, les plans sont moins stables lorsque l'on part d'un plan sans action hiérarchique. Cela vient du fait que plusieurs essais sont nécessaires pour réparer un plan. A chaque nouvel essai, on retire plus d'éléments du plan. S'il faut deux essais pour trouver une solution, on aura retiré plus d'actions du plan.

On voit donc que le fait de produire des plans hiérarchiques (donc avec une information supplémentaire dans le plan) permet de faciliter la réparation. Les actions abstraites permettent de déterminer pourquoi une action élémentaire a été ajoutée dans le plan et cela permet de mieux cibler quelles sont les actions à retirer pour pouvoir réparer le plan.

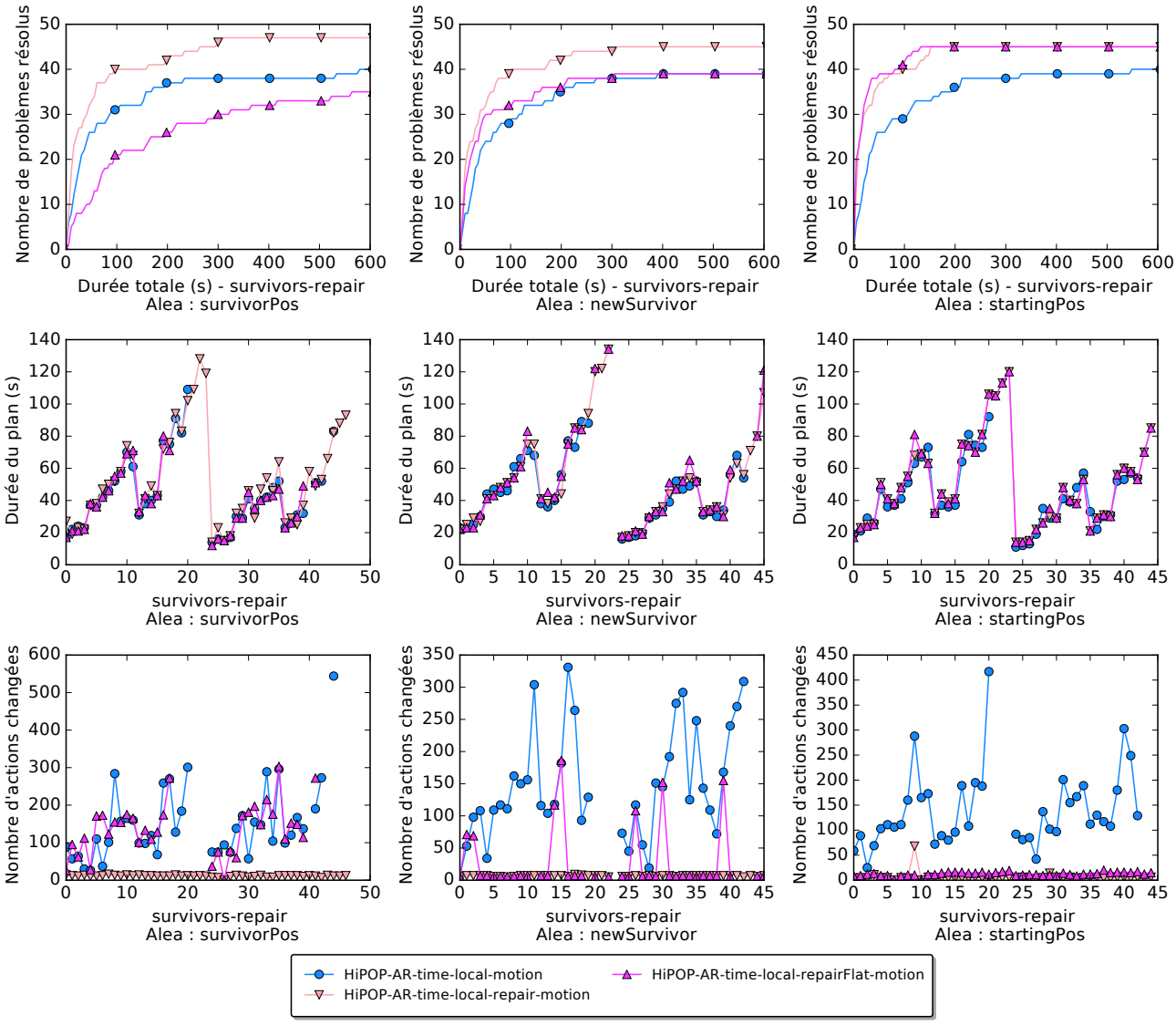


Figure 5.12 – Comparaison de plusieurs planificateurs sur le nombre de problèmes résolus, la durée du plan trouvé et la stabilité du plan. Chaque colonne correspond à un aléa différent. La première ligne présente le nombre de problèmes qui ont été résolus en utilisant au plus le temps indiqué en abscisse. La deuxième ligne présente la durée de chaque plan trouvé. La troisième ligne présente la stabilité de chaque plan : le nombre de tâches ayant été enlevées ou rajoutées entre le plan trouvé et le plan initial.

Conclusion

Dans ce chapitre, nous avons développé un algorithme de réparation adapté de l'algorithme de planification hybride HiPOP.

Cet algorithme calcule itérativement un plan initial qui est ensuite fourni à HiPOP comme point de départ de sa recherche. Si aucune solution n'est trouvée, il retire de plus en plus d'éléments du plan jusqu'à avoir retiré tous les éléments possibles. Les liens causaux et les actions hiérarchiques sont utilisés pour déterminer quels sont les éléments pertinents

à retirer. Le but est de produire un plan solution du problème le plus proche possible du plan initial. De plus, certains éléments indispensables (comme les buts à atteindre ou les tâches déjà exécutées) sont garantis d'être présents dans le plan produit.

Des améliorations ont été proposées pour permettre à cet algorithme de faire la différence entre les tâches qu'il peut ajouter ou non dans le plan (pour ne pas ajouter de tâches à un robot avec lequel il n'est pas possible de communiquer) ou pour permettre de produire un plan résolvant le plus de buts possibles (au lieu de produire un échec si un seul des buts est inatteignable).

On a vérifié expérimentalement, dans le cadre d'une planification hors ligne (donc sans action en cours d'exécution), que la réparation était plus performante qu'une replanification. Elle permet de réduire le temps de recherche et de garder une certaine stabilité du plan sans en augmenter significativement la durée. On a aussi montré l'apport de la présence d'actions hiérarchiques dans le plan pour la réparation. Il n'a néanmoins pas été possible de vérifier expérimentalement ses capacités sur des plans en cours d'exécution en l'absence d'un algorithme d'exécution.

Dans le prochain chapitre, nous allons donc nous intéresser à l'exécution d'un plan hybride. Cela permettra de réaliser des missions comprenant la planification initiale de la mission, l'exécution du plan et la réparation éventuelle en cours de mission.

6

Intégration d'HiPOP dans une architecture d'exécution multiagent

Sommaire

Le but de ce chapitre est d'avoir une architecture logicielle permettant la réalisation de missions de surveillance de zone par une équipe de robots autonomes hétérogènes en environnement réel extérieur. Les chapitres précédents ont montré comment on pouvait produire des plans hybrides (*i.e.* flexibles temporellement et possédant une hiérarchie parmi les actions) pour résoudre un problème donné et comment on pouvait réparer un plan en cours d'exécution. Il est maintenant nécessaire d'intégrer ce planificateur dans une architecture embarquée à bord de robots pour réaliser une mission.

Dans ce qui suit, on suppose que chaque robot a une certaine autonomie et est capable de réaliser seul des actions de déplacement, d'observation, etc. Chaque robot peut inclure un autre planificateur et/ou superviseur pour exécuter ces actions, éventuellement différents sur chaque robot. On suppose seulement de notre point de vue qu'il est possible de lancer l'exécution de ces actions sur chaque robot et qu'un retour est fourni concernant le succès (ou non) de l'action et sa date de fin. L'échec d'une action correspond alors à un événement perturbateur : obstacle sur la route, cible détectée, panne, etc.

Dans ce chapitre, nous allons donc nous intéresser dans un premier temps à l'algorithme de supervision du plan. Son rôle est de déterminer à tout instant, en fonction du plan et du retour des actions précédentes, quelles actions doivent être réalisées à bord de chaque robot. Il est aussi responsable de l'appel au planificateur pour réparer le plan pendant l'exécution. Nous nous intéresserons ensuite à la modélisation des missions exécutées dans le cadre du Programme d'Études Amont (PEA) ACTION. Cela concerne à la fois la description des actions hiérarchiques disponibles et le comportement souhaité des robots lors de la survenue d'un événement perturbateur. Finalement, des résultats seront présentés sur des missions simulées puis sur des missions en environnement réel.

6.1 Supervision distribuée de plans hybrides

Le rôle de la supervision est de superviser l'exécution du plan : exécuter les actions au bon moment, vérifier la cohérence du plan avec les différentes observations et réagir en cas d'occurrence d'un événement perturbateur.

Dans un contexte multirobot, chaque robot supervise son propre plan et il est nécessaire de synchroniser les différents robots durant tout la durée de l'exécution du plan. Dans le cadre de ces travaux, nous avons développé un algorithme d'exécution nommé METAL (Multiagent Execution with TemporAl fLexibility). Cet algorithme est déployé sur chaque robot pour superviser les actions du robot et est capable de communiquer avec les superviseurs des autres robots pour synchroniser l'exécution du plan avec les autres robots. Ainsi, même en l'absence de communication, l'exécution du plan sur chaque robot peut continuer.

6.1.1 Supervision monoagent d'un plan flexible temporellement

Dans un premier temps intéressons-nous à l'exécution monoagent d'un plan flexible temporellement (Algorithme 4 ci-après).

Chaque tâche dans le plan est associée à un instant temporel de début et un instant temporel de fin sur lesquels sont définies un ensemble de contraintes temporelles. Ces contraintes sont issues des liens causaux ou des liens temporels du plan (lors de l'exécution, on ne fait pas de différence entre ces deux contraintes). De plus, chaque instant temporel peut être contrôlable ou incontrôlable du point de vue de la supervision. Un instant contrôlable peut être déclenché par la supervision alors que la survenue d'un instant incontrôlable est due à un événement extérieur. Toutes les contraintes temporelles sont stockées dans un STN.

Pour la majorité des actions élémentaires, l'instant initial est contrôlable (*i.e.* le superviseur peut déclencher l'action) mais sa fin est incontrôlable (*i.e.* quand le robot a fini son action il en informe le superviseur). Les actions abstraites sont associées à des instants contrôlables : le superviseur décide quand une action abstraite commence et termine. Ces actions ne peuvent pas être exécutées par les robots, mais elles sont gardées dans le plan pour faciliter la réparation. Certaines actions élémentaires peuvent aussi être totalement contrôlables. Par exemple une action d'observation ou d'attente de communication peut être terminée par le robot (si quelque chose a été détecté) ou par le superviseur (pour continuer l'exécution du plan).

Un instant peut être exécuté par le superviseur si toutes les contraintes temporelles sont satisfaites (ligne 7), *i.e.* si tous les instants temporels précédents sont déjà exécutés et si la date courante est bien comprise entre les dates d'exécution au plus tôt (ligne 8, noté *lb* pour *lower bound*) et au plus tard (ligne 9, noté *ub* pour *upper bound*) de l'instant dans le STN. A chaque exécution d'un instant du STN, si cet instant est le début ou la fin d'une action élémentaire, le superviseur donne l'ordre au robot de commencer ou de terminer l'action (ligne 14).

Lorsque le superviseur est notifié de la survenue d'un instant temporel (comme la fin d'une action), il met à jour le STN avec la date de l'événement. Le compte rendu d'exécution d'une action (ligne 16) contient le nom de l'instant incontrôlable qui a été exécuté (*tp*), sa date d'exécution (t_e) et son statut. Le statut peut indiquer que l'action s'est correctement déroulée ou qu'un problème a été rencontré.

Mettre à jour le STN consiste à propager la nouvelle contrainte temporelle dans tout le STN pour recalculer les dates d'exécution possibles des points et à vérifier qu'il est toujours cohérent. Des algorithmes efficaces ont été proposés dans la littérature pour

Algorithme 4 : Algorithme de supervision

Entrée : STN, le STN à exécuter
start l'instant de départ
end l'instant final

```
1  E = {start}                                   // Ensemble des points exécutés
2  tant que end ∉ E faire
3      Soit t la date actuelle
4
       // Exécution de points
5      pour chaque tp ∈ STN faire
6          si tp ∈ E ou tp est incontrôlable alors continuer
7          si tp a des préconditions non exécutées alors continuer
8          si lb(tp) < t alors continuer
9          si ub(tp) > t alors retourner échec
10         sinon
11             E = E ∪ {tp}
12             Mise à jour du STN avec lb(tp) = ub(tp) = t
13             si tp est le début d'une action élémentaire alors
14                 Exécuter l'action
15
       // Compte rendu d'exécution
16     pour chaque compte rendu d'action {tp, status, t_e} faire
17         si status = échec alors
18             retourner échec
19         sinon
20             E = E ∪ {tp}
21             Mise à jour du STN avec lb(tp) = ub(tp) = t_e
22             si STN est incohérent alors
23                 retourner échec
24 retourner succès
```

maintenir de manière incrémentale un STN [Cesta et Oddi 1996 ; Planken, Weerdt et Yorke-Smith 2010]. La notification de la fin d'une action peut éventuellement faire progresser le plan en permettant l'exécution de l'action suivante dans le plan.

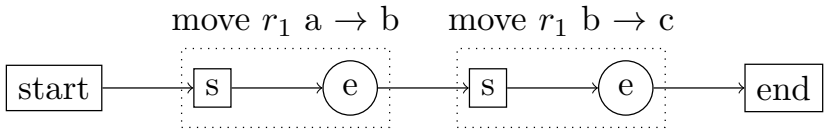
Si le STN devient incohérent durant l'exécution (par exemple parce qu'une action en retard empêche de respecter une échéance), alors il est nécessaire de réparer le plan. Le plan actuel est fourni au planificateur avec la définition du problème à résoudre (cf. chapitre 5). Une fois le plan réparé, si une solution est trouvée, l'exécution peut reprendre.

L'algorithme présenté ici (Algorithme 4) itère sur tous les points du STN à chaque étape. On pourrait réduire cette complexité en maintenant en permanence la liste des instants non exécutés et dont toutes les préconditions sont exécutées. Cette liste serait mise à jour à chaque exécution de point (lignes 11 et 20) et il suffirait d'itérer sur cette liste au lieu d'itérer sur tous les points du STN. En pratique, cette amélioration n'a pas été nécessaire au vu des temps de calcul faibles.

Exemple : Exécution monoagent d'un plan flexible temporellement

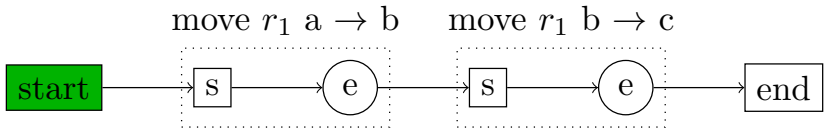
On suppose ici que l'on souhaite exécuter un plan composé de deux actions pour un robot unique. Le plan initial est représenté sur la figure 6.1a.

Figure 6.1 – Exemple d'exécution d'un plan monorobot.



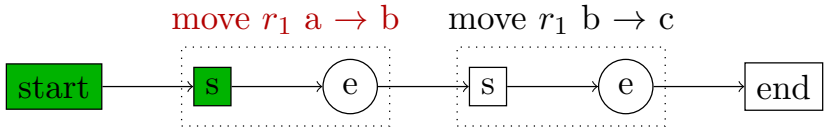
(a) Chaque instant temporel est représenté par un nœud rectangle s'il est contrôlable et rond s'il est incontrôlable. Chaque action est représentée par un rectangle en pointillé comprenant ses instants initiaux (s) et finaux (e). Les flèches représentent les contraintes de précédence temporelle.

Au lancement de l'exécution du plan, on exécute l'instant initial du plan. On aboutit alors au plan suivant :



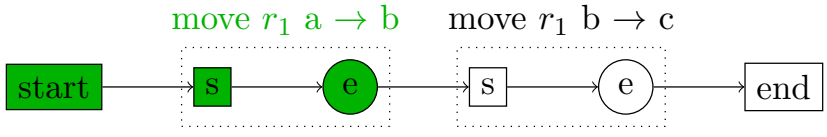
(b) Plan en cours d'exécution. Les instants exécutés sont représentés en vert.

L'instant de départ de la première action est alors réalisable : il est contrôlable et tous les instants précédents sont exécutés. On peut donc lancer son exécution, ce qui revient à envoyer l'ordre de déplacement au robot r_1. On aboutit alors au plan suivant :



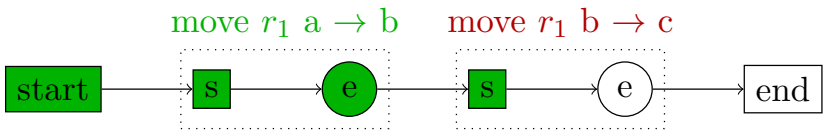
(c) Plan en cours d'exécution. Les instants exécutés sont représentés en vert. Les actions futures sont représentées en noir, les actions en cours d'exécution en rouge et les actions exécutées en vert.

A ce moment-là aucun instant n'est exécutable : ils dépendent tous de la fin de l'action en cours d'exécution qui est incontrôlable. Il faut alors attendre la fin de l'action en cours. Lorsque le robot finit son action, il en informe le superviseur. On arrive alors à ce plan :



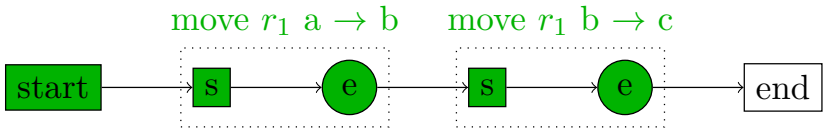
(d) Plan en cours d'exécution. Les instants exécutés sont représentés en vert. Les actions futures sont représentées en noir, les actions en cours d'exécution en rouge et les actions exécutées en vert.

Maintenant que l'action est terminée, elle est indiquée dans le plan comme exécutée. L'exécution peut alors continuer comme précédemment : l'instant de début de l'action suivante devient exécutable.



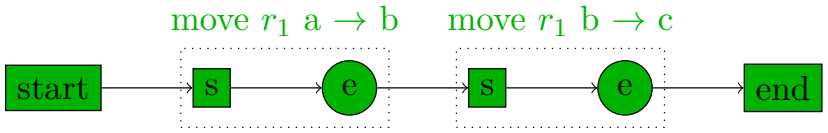
(e) Plan en cours d'exécution. Les instants exécutés sont représentés en vert. Les actions futures sont représentées en noir, les actions en cours d'exécution en rouge et les actions exécutées en vert.

On attend alors la fin de l'action pour aboutir à :



(f) Plan en cours d'exécution. Les instants exécutés sont représentés en vert. Les actions futures sont représentées en noir, les actions en cours d'exécution en rouge et les actions exécutées en vert.

A ce moment-là l'instant final devient exécutable. Cela signale que toutes les actions ont été finies et que le but est atteint : c'est la fin du plan.



(g) Plan complètement exécuté.

Tant qu'une action n'est pas exécutée, on utilise comme durée de l'action (celle utilisée dans le STN et lors de la planification) l'estimation qui est faite dans le problème. Une fois que l'action termine, on met à jour cette durée avec la durée réelle (qui peut être plus courte ou plus longue).

6.1.2 Supervision distribuée

Lorsque le plan à exécuter concerne plusieurs agents, diverses adaptations sont nécessaires.

Chaque robot étant indépendant, il possède son propre superviseur. En cas de perte de communication par exemple, chaque robot doit pouvoir continuer l'exécution de son plan. Ce superviseur est responsable des actions du robot, mais pas des actions des autres robots.

Le chapitre précédent introduit la définition du responsable de chaque action dans la description des actions (cf. section 5.3.1). Lors de l'exécution, le superviseur d'un robot ne peut contrôler que les actions de ce robot. Tous les instants temporels appartenant aux tâches des autres robots sont considérés comme incontrôlables.

Il est alors nécessaire que chaque superviseur informe les autres superviseurs quand un de ses instants est exécuté pour permettre de synchroniser les exécutions.

Néanmoins, toutes ces informations ne sont pas utiles à envoyer. Certains instants ont des contraintes les liant avec des instants d'autres robots : il est nécessaire de les partager. Mais la majorité des instants ne sont pertinents que pour un seul robot et n'ont pas de contraintes avec les actions des autres robots.

Pour limiter la quantité d'information envoyée entre les superviseurs, nous avons donc utilisé un MaSTN (MultiAgent Simple Temporal Network) particulier [Casanova, Pralet et Lesire 2015].

L'idée est que chaque robot va utiliser un STN différent appelé MacroSTN. Ce MacroSTN reprend tous les nœuds du STN qui concerne directement le robot plus tous les nœuds qui sont en relation avec des nœuds d'au moins 2 robots. Ces nœuds partagés composent le minimum nécessaire pour permettre aux robots de se synchroniser : les informations concernant ces nœuds permettent de déclencher les actions des autres robots ou de mettre à jour les dates au plus tôt et au plus tard des actions qui dépendent des autres robots. Les informations échangées entre les robots sont donc uniquement celles qui concernent le MacroSTN et pas toutes celles concernant le STN général. Dans la pratique, ces informations sont échangées dès qu'une action commence ou termine (*i.e.* dès qu'une nouvelle information est disponible) et toutes les 30 secondes (pour être robuste à la perte de certains messages). Dans la mission qui sera présentée à la section 6.4, on passe d'un STN avec 256 instants temporels à un MacroSTN avec 82 instants pour un agent.

Une dernière modification concerne l'instant final du plan. L'instant initial reste le même pour tous les robots : le début de la mission est centralisé et tous les robots reçoivent l'ordre de départ ensemble. Mais l'instant final est différent : chaque robot peut finir sa partie du plan à un instant différent. On a donc défini une fin pour chaque robot, correspondant à un instant postérieur à toutes les actions d'un robot donné. La fin globale du plan correspond elle à un instant postérieur à toutes les fins des robots individuels. C'est l'exécution de la fin globale qui signale la fin de la mission pour l'ensemble des robots.

Cela nous amène donc à définir deux formes de plans différentes.

- **Le plan global** correspond au plan contenant toutes les actions, tel que calculé par le planificateur. Ce plan est associé à un MaSTN.

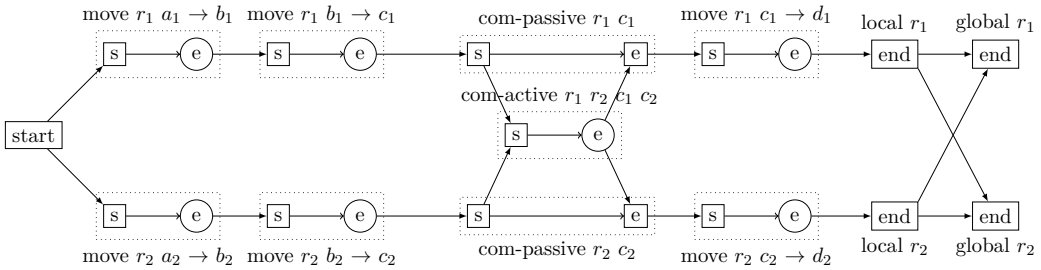
- **Le plan local** à un robot correspond au plan contenant uniquement les tâches du robot et les instants temporels nécessaires à la synchronisation (*i.e.* ceux étant liés par une contrainte temporelle à un instant appartenant au robot). Il est associé à un MacroSTN et est extrait d'un plan global.

Dans l'algorithme 4, les changements sont mineurs. Le STN est remplacé par un MacroSTN et tous les points appartenant à un autre agent sont incontrôlables. Lors de la mise à jour du MacroSTN (lignes 12 et 21), les contraintes temporelles nécessaires à la synchronisation des MacroSTN sont envoyées aux autres robots.

Exemple : Exécution multirobot d'un plan flexible temporellement

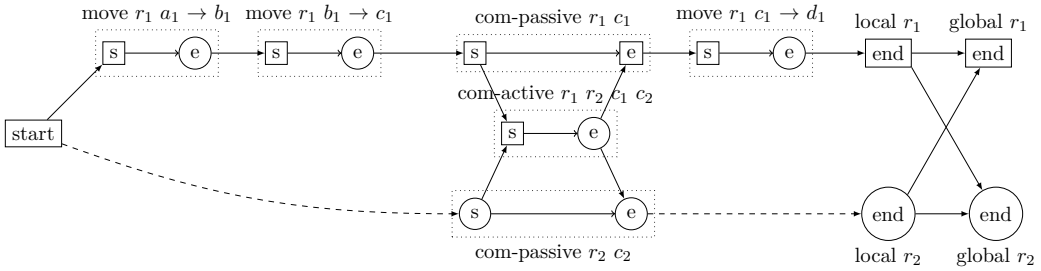
Supposons que l'on cherche à exécuter le plan suivant :

Figure 6.2 – Exemple d'exécution d'un plan multirobot.



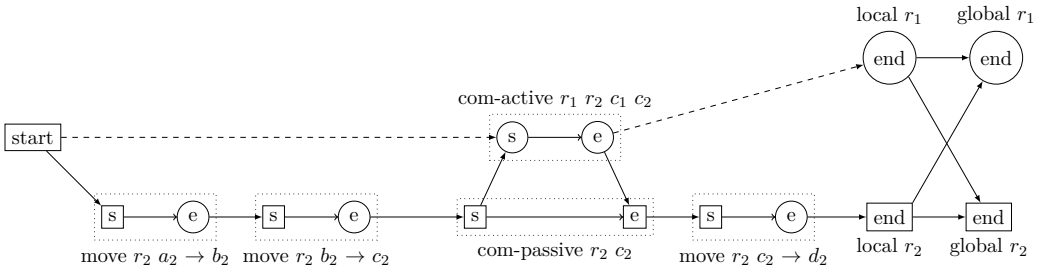
(a) Plan global initial. Chaque instant temporel est représenté par un nœud rectangle s'il est contrôlable et rond s'il est incontrôlable. Chaque action est représentée par un rectangle en pointillé comprenant ses instants initiaux (s) et finaux (e). Les flèches représentent les contraintes de précédence temporelle.

Ce plan concerne deux robots. Chaque robot a deux actions de déplacement à réaliser avant un rendez-vous planifié en `c`. Après ce rendez-vous, il reste une action à effectuer pour chaque robot. On suppose de plus que le robot r_1 est responsable de l'action `com-active`. Les contraintes temporelles forcent cette action à avoir lieu une fois que les deux robots sont prêts.
La fin du plan a été scindée en plusieurs instants : chaque robot possède un instant représentant la fin de son plan (appelé *local*) et un autre représentant la fin de la mission (appelé *global*).
Du point de vue de r_1, le MacroSTN permet de simplifier cette représentation. Les actions de r_2 sont remplacées par des contraintes temporelles externes : leur valeur peut évoluer en cours d'exécution mais ne dépend pas du superviseur. Les instants de r_2 possédant des contraintes avec des points de r_1 sont gardés en tant qu'instants incontrôlables. Ce plan est le suivant :



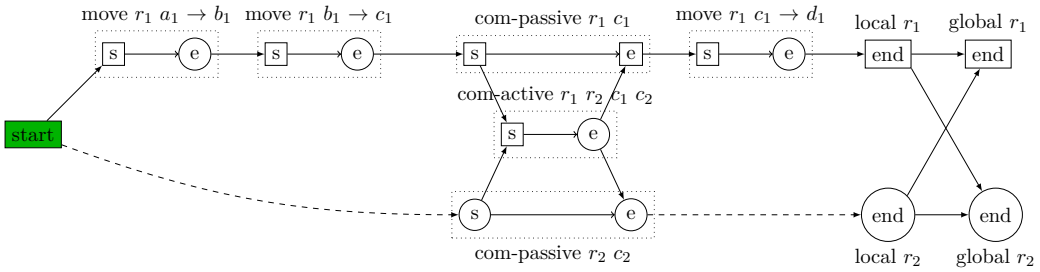
(b) Plan local initial pour r_1. Les contraintes extérieures sont représentées en pointillé.

Du coté de r_2, son plan local est le suivant :



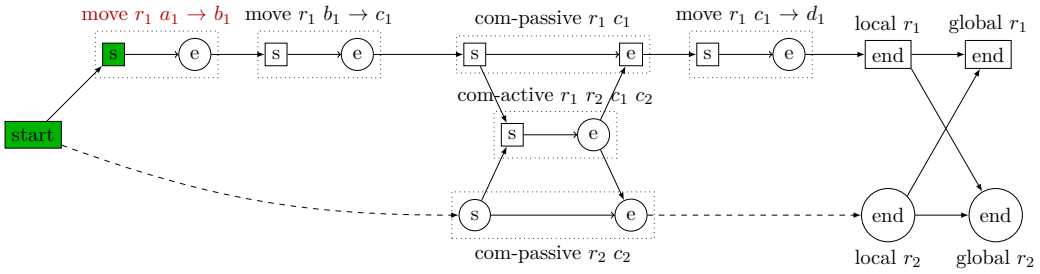
(c) Plan local initial pour r_2. Les contraintes extérieures sont représentées en pointillé.

Suivons le robot r_1. Lorsque le plan débute, l'instant initial est exécuté :



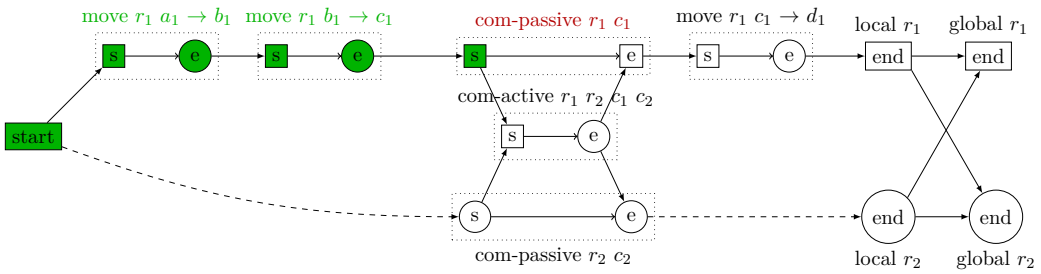
(d) Plan local, en cours d'exécution, pour r_1. Les contraintes extérieures sont représentées en pointillé. Les instants exécutés sont représentés en vert.

Il rend donc possible l'exécution de la première action de déplacement :



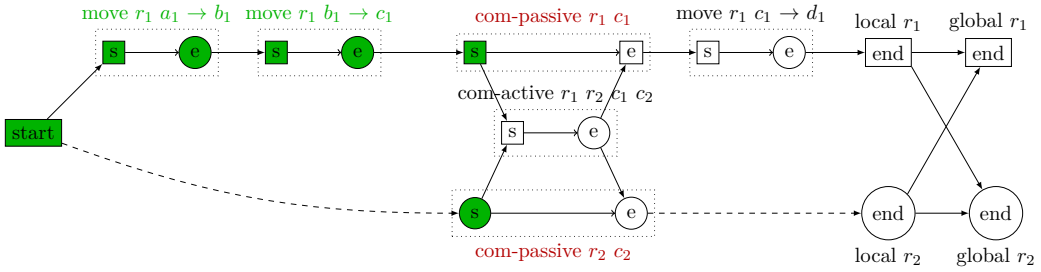
(e) Plan local, en cours d'exécution, pour r_1. Les contraintes extérieures sont représentées en pointillé. Les instants exécutés sont représentés en vert.

Lorsque l'autre robot exécute ses propres actions, il met à jour la contrainte entre `start` et le début de l'action `com-passive`. Si le robot r_1 finit ses actions avant r_2, on aboutit au plan suivant :



(f) Plan local, en cours d'exécution, pour r_1. Les contraintes extérieures sont représentées en pointillé. Les instants exécutés sont représentés en vert.

Le robot r_1 attend alors la survenue d'un point incontrôlable pour lui (mais contrôlable par r_2) qui représente le fait que le robot r_2 est prêt pour la communication. Une estimation de la date de survenue de cet instant est donnée par la contrainte temporelle issue et envoyée régulièrement par le robot r_2.
Une fois cet instant exécuté par r_2, r_1 en est informé. On aboutit alors au plan suivant :



(g) Plan local, en cours d'exécution, pour r_1. Les contraintes extérieures sont représentées en pointillé. Les instants exécutés sont représentés en vert

Dans cette situation, l'action `com-active` devient réalisable. r_1 peut donc lancer la communication (échange de données, etc.). Quand toutes les données ont été transférées, l'action se termine. r_1 reprend alors l'exécution de son plan et en notifie r_2 qui peut lui-même continuer.
La fin du plan est identique : chaque robot continue son plan jusqu'à exécuter la fin locale de son plan. Une fois que toutes les fins locales ont été exécutées, les robots s'en informent et ils peuvent tous exécuter leur fin globale.

6.1.3 Réparation distribuée

Durant l'exécution, plusieurs aléas peuvent survenir. Les retards des actions sont pris en compte lors de l'exécution : tous ceux qui ne rendent pas une échéance impossible à tenir sont gérés par l'algorithme de supervision. Mais certains aléas ne peuvent pas être traités par la supervision. C'est pourquoi il est nécessaire de réparer le plan.

Comme on l'a vu au chapitre 5, l'algorithme de planification utilisé permet de réparer un plan. Mais il n'agit que sur un plan global, il ne peut réparer un plan local car un tel

plan ne contient pas toutes les actions.

Il est donc nécessaire de calculer le plan global avant de pouvoir réparer. Cette opération consiste à récupérer tous les plans locaux des différents agents, avec leurs informations d'exécution précises comme les dates de réalisation de toutes les actions, et à les rassembler pour en faire un plan global.

Ce plan global est ensuite réparé et distribué aux robots : on recalcule un plan local pour chaque robot en fonction du plan global réparé. Comme les informations d'exécution étaient dans le plan global à réparer et qu'elles ont été conservées par le processus de réparation, elles se retrouvent dans les nouveaux plans locaux. Un agent se retrouve donc à exécuter un plan local cohérent avec son plan local antérieur.

En cas de communications coupées, certains robots peuvent ne pas être joignables. Dans ce cas, ils ne participent pas à la réparation. Le plan global est calculé en fonction du dernier plan local connu des robots hors de portée de communication. Mais la réparation ne doit ni ajouter ni retirer des actions aux robots qui ne sont pas à portée de communication (cf. section 5.3.1).

Ce processus de fusion des plans n'est possible que si les plans locaux sont issus du même plan global. Il est donc nécessaire de garantir que les robots exécutent le même plan global, même en présence de pertes de communication. C'est ce qu'on appelle la synchronisation des plans.

A chaque message envoyé on ajoute un identifiant unique du plan global en cours d'exécution ainsi que l'identifiant de tous les plans ayant précédé. Ces identifiants permettent de déterminer quel plan est issu de quel autre plan et de vérifier que tous les robots exécutent le même plan. Si un robot détecte qu'un autre robot envoie un identifiant différent du sien, cela signifie qu'une réparation a eu lieu lors de laquelle tous les robots n'étaient pas présents. Il est alors nécessaire que chaque robot mette à jour son plan : chaque robot commence par envoyer son plan global.

Supposons que le robot r reçoive un plan global d'un autre robot. Si l'autre robot exécute le même plan ou un plan plus ancien, aucune modification n'est nécessaire. Si l'autre robot a un plan plus récent, alors il est nécessaire de l'utiliser comme plan global. Comme la réparation des autres robots s'est faite sans r, son plan local dans l'ancien plan global doit être le même que son plan local dans le nouveau plan global. r peut alors insérer son plan local actuel dans le nouveau plan global. Le dernier cas de figure est que deux robots exécutent deux plans dont aucun n'est issu l'un de l'autre : cela signifie qu'il y a eu deux réparations distinctes. Dans ce cas, une nouvelle réparation est nécessaire.

Exemple : Réparation multirobot

Supposons que l'on exécute une mission avec 3 robots. À un moment donné, une réparation est nécessaire alors que le robot 3 est hors de portée de communication. Après la réparation, la communication est rétablie. L'évolution des plans est représentée sur la figure 6.3.
La première colonne, *Exécution*, correspond à une exécution classique : tous les robots ont le même plan global. Chacun exécute sa partie du plan et informe les autres de l'avancée de son exécution.
La deuxième colonne, *Fusion*, correspond au début du processus de réparation. Le problème a été détecté par le premier robot, qui a contacté les autres robots pour récupérer leur plan local. Comme il ne peut pas communiquer avec le troisième

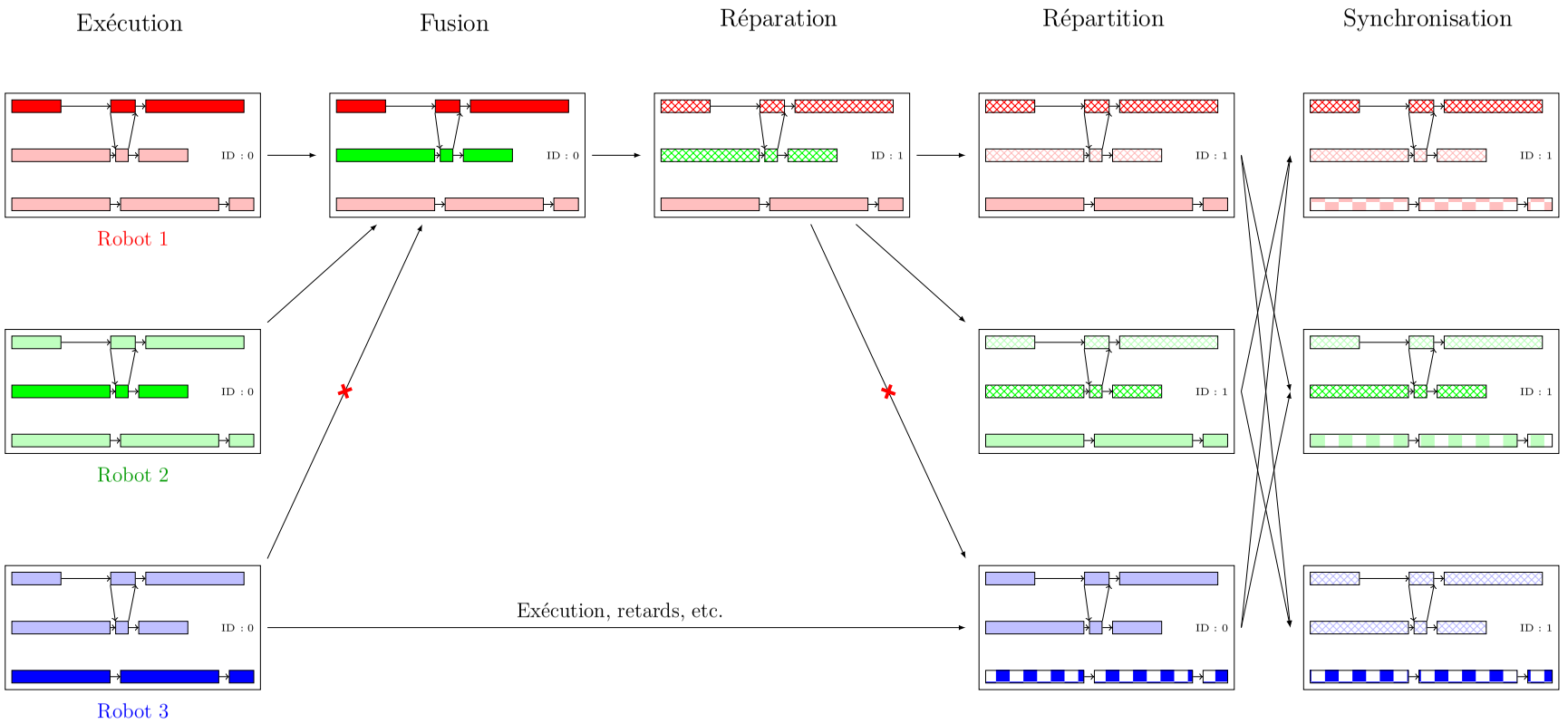


Figure 6.3 – Exemple d'un processus de réparation distribuée. Chaque ligne correspond au plan global d'un robot donné. Chaque ligne d'un plan correspond au plan local d'un robot donné.

robot, il utilise ses dernières informations connues dans le plan et s'interdira de les modifier pendant la réparation.
La troisième colonne, *Réparation*, correspond au processus de réparation d'un plan global par HiPOP. La partie concernant le troisième robot est intouchée, seules les actions des robots 1 et 2 sont modifiées. Ces actions modifiées sont indiquées sur la figure par des croisillons.
La quatrième colonne, *Répartition*, correspond à l'envoi du nouveau plan à tous les robots. Le robot 3 est toujours hors de portée de communication donc ne reçoit rien, mais le plan est correctement pris en compte par le robot 2. Pendant ce temps, le troisième robot continue son exécution. Il met donc à jour les dates d'exécution de ses actions. Cela est représenté par les actions en damier dans le plan. A ce moment-là, les robots exécutent des plans avec des identifiants différents. Mais tant que la communication n'est pas rétablie, ils ne peuvent pas s'en rendre compte.
La cinquième colonne, *Synchronisation*, correspond à ce qui se passe une fois que la communication est rétablie (ce qui peut arriver bien plus tard que les autres événements). Les robots détectant les deux plans différents, ils partagent tous leur plan global. Les robots 1 et 2 ont des plans plus récents : ils ne changent pas leurs plans mais utilisent les nouvelles informations d'exécution du robot 3. Le robot 3 se rend compte qu'il a un plan obsolète. Il met donc à jour son plan global en fusionnant le plan issu des autres robots et son plan local actuel. On se retrouve alors avec des plans synchronisés : tous les robots exécutent le même plan global.

6.1.4 Machine à état interne

Pour que la réparation telle que présentée ci-dessus soit valide, il ne faut pas que le plan évolue pendant la réparation. En particulier il ne faut pas lancer l'exécution d'actions pendant la réparation : le plan n'est plus valide et le plan réparé ne les contiendra pas forcément.

Pour garantir cela, l'état du superviseur de chaque robot est représenté sous forme d'un état dans une machine à état. Cette machine à état est représentée sur la figure 6.4.

Les robots commencent dans l'état *Init* : ils attendent l'ordre de commencer la mission. Lorsque l'opérateur donne l'ordre, les robots passent dans l'état *Running* (et le premier instant temporel du STN est exécuté). C'est le seul état dans lequel des nouvelles actions peuvent être exécutées. Si un robot détecte un problème nécessitant une réparation, il passe dans l'état *Repair active* signifiant qu'il est le robot qui répare. Les autres robots à portée de communication passent dans l'état *Repair passive* et attendent la réparation du plan. Les autres états particuliers du plan sont *Done* (la fin globale du plan est exécutée, la mission est finie), *Dead* (le robot a connu un dysfonctionnement et ne peut plus rien faire) et *Error* (si la réparation échoue, il n'est plus possible de continuer la mission).

Une transition particulière est celle qui permet de passer de l'état *Repair active* à l'état *Repair passive*. Ce cas ne se présente que si plusieurs réparations sont lancées en simultané : on a alors plusieurs robots dans l'état *Repair active*. Ce cas se présente par exemple quand un robot envoie une contrainte temporelle qui rend le STN inconsistant : plusieurs robots vont la recevoir et lancer une réparation en même temps. Quand un robot dans l'état *Repair active* reçoit une demande de réparation d'un autre robot (signifiant que l'autre robot est lui aussi passé dans l'état *Repair active*), le robot ayant la plus faible priorité

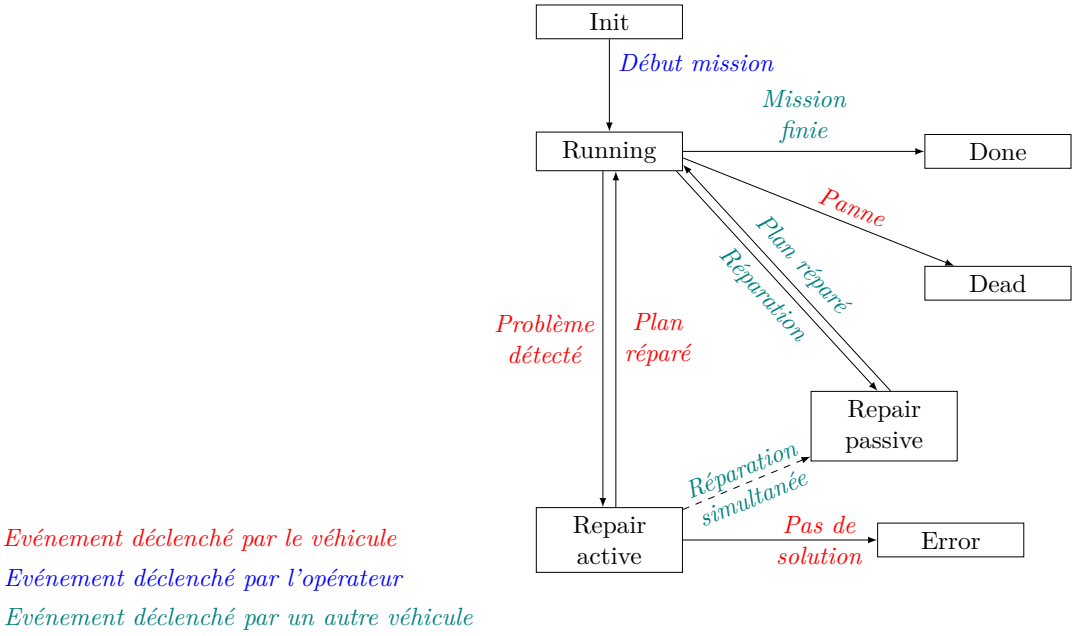


Figure 6.4 – Machine à état interne de METAL. Les transitions sont classées en 3 types : celles qui sont causées par le robot lui-même, celles causées par un autre robot et celles causées par l'opérateur.

passe dans l'état *Repair passive*. Dans notre cas, la priorité utilisée est l'ordre alphabétique mais cette priorité pourrait prendre en compte les capacités de calcul de chaque robot par exemple. Ce mécanisme garantit qu'un seul robot répare à la fois.

6.1.5 Suivi de l'exécution du plan

Pour permettre de suivre l'exécution de la mission, deux interfaces graphiques différentes ont été développées pour METAL.

La première, nommée Timeline, permet de suivre l'état d'exécution du plan par chaque robot. Chaque robot envoie périodiquement son plan local, qui est affiché à l'opérateur. Une capture d'écran est présentée à la figure 6.5.

Pour l'opérateur, une interface graphique a été créée pour suivre la position géographique des robots et permettre d'envoyer toutes les consignes à sa disposition. Cette interface s'appelle ISMAC (Interface de Supervision de Mission pour ACtion). Une capture d'écran est présentée sur la figure 6.6.

Cette interface permet de lancer la mission, de déclarer des robots inopérationnels, de simuler des pertes de communications, etc.

6.2 Intégration dans le PEA ACTION

Ce travail a été intégré dans les scénarios aéroterrestres du Programme d'Études Amont (PEA) ACTION, ce qui a permis de le tester à la fois en simulation et en expérimentation en environnement réel.

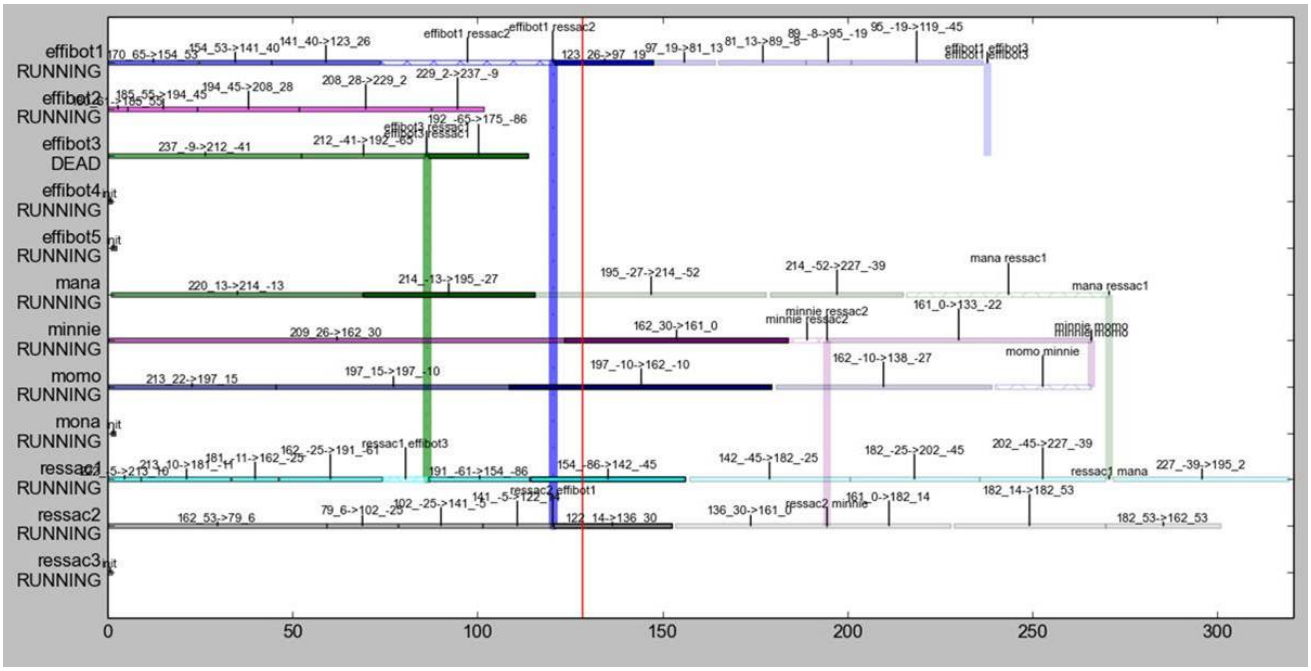


Figure 6.5 – Exemple de capture d'écran de Timeline. Chaque ligne correspond à un robot et indique l'état dans lequel il est. La barre rouge verticale correspond à la date actuelle. Les actions verticales correspondent aux actions de communication : ici deux communications ont été effectuées et quatre sont à venir. Les actions les plus foncées représentent l'action en cours d'exécution par un robot.

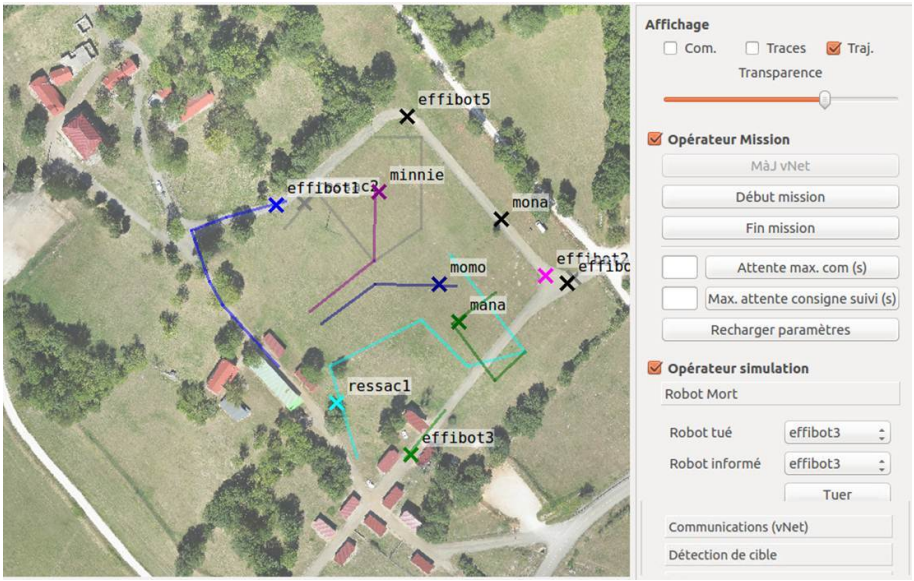


Figure 6.6 – Exemple de capture d'écran de l'interface de supervision ISMAC.

6.2.1 Présentation du PEA ACTION

L'objectif du PEA ACTION [1] est de développer et d'implémenter sur des véhicules hétérogènes autonomes une architecture logicielle multidrone permettant de les faire coopérer pour la réalisation de leur mission.

Les scénarios aéroterrestres impliquent des robots terrestres et aériens dans une mission de contrôle de zone : le but est de patrouiller dans une zone connue à la recherche de cibles éventuelles. Si une cible est détectée, on suppose qu'elle a un comportement évasif. Le but est alors de la suivre jusqu'à ce qu'elle sorte de la zone et ce but est prioritaire par rapport à l'exploration de la zone.

Le projet est constitué de plusieurs scénarios de complexité croissante. Dans le scénario V, 4 robots sont déployés (2 AGV et 2 AAV). Dans le scénario VI, 12 robots sont déployés (9 AGV et 3 AAV).

Les robots utilisés lors du projet sont :

- Les AAV ReSSAC de l'ONERA (Office national d'études et de recherches aérospatiales, deux exemplaires, cf. figure 6.7a) ;
- Les AGV Mana, Minnie et Momo du LAAS (Laboratoire d'analyse et d'architecture des systèmes du CNRS, cf. figure 6.7b) ;
- Les AGV Effibot de la DGA (Direction générale de l'Armement), mis en œuvre par l'ONERA (cf. figure 6.7c).
- Des robots simulés ayant les mêmes capacités que ces robots réels (cf. figures 6.7d et 6.7e).

Tous les robots possèdent une couche autonome permettant de rallier des points de passage. Les AAV ReSSAC peuvent suivre une cible équipée d'un émetteur GPS. Les AGV du LAAS peuvent détecter et suivre une cible munie de bandes réfléchissantes. Pour compléter l'équipe de robots réels, des robots simulés sont utilisés durant certaines démonstrations. Les simulations mettant en œuvre à la fois des robots réels et des robots simulés sont appelés *simulations hybrides*. Ces simulations sont faites avec le simulateur robotique MORSE [Echeverria et al. 2011]. Les communications entre superviseurs sont réalisées à travers le middleware ROS (Robot Operating System). Les expérimentations ont été menées sur le site du camp militaire de Caylus (Tarn-et-Garonne).

6.2.2 Modélisation de la mission

Chaque type de robot se voit assigner un ensemble de points de passage accessibles. Un modèle numérique du terrain est utilisé pour calculer la durée probable du déplacement de chaque robot entre chaque paire de points de passage.

L'objectif de la mission est modélisé par un ensemble de points à explorer. Le modèle numérique du terrain est utilisé pour déterminer de quels points de passage sont visibles les points à observer pour chaque type de robots.

La taille du terrain choisi pour les essais étant limitée par rapport au nombre de robots, il n'a pas été possible de la séparer en plusieurs zones auxquelles auraient été assignées des équipes distinctes, comme dans le domaine *survivors* utilisé jusqu'ici. Les actions hiérarchiques prennent donc la forme de patrouilles pour un robot unique, chaque patrouille

1. `http://action.onera.fr`

(a) AAV ReSSAC de l'ONERA.

(b) AGV du LAAS.

(c) AGV Effibot de la DGA.

(d) Véhicule simulé (AGV) sous MORSE.

(e) Véhicule simulé (AAV) sous MORSE.

Figure 6.7 – Les différents robots participant au projet ACTION.

étant définie par un ensemble de points de passage ordonnés. Ces patrouilles sont aussi définies pour respecter les contraintes dynamiques des robots : limitation des virages trop serrés pour les AAV, suivi des chemins pour certains robots, etc.

Pour atteindre l'objectif d'une équipe de 12 robots, 4 robots simulés sont venus compléter les 8 robots disponibles. Ces robots simulés sont considérés comme des robots de remplacement : leur plan initial est vide mais ils font partie de l'équipe, ils sont disponibles en cas de réparation.

Pour faciliter la description de la mission par l'opérateur, une interface de préparation a été développée. Cette interface est appelée SIMATO (Specification Interface for Multi-Agents Tactical Operations) et une capture d'écran est présentée sur la figure 6.8. L'opérateur choisit ainsi les robots et leur type, la zone de mission et son image de fond, les sous-zones dans lesquelles sont contraints les robots aériens (contrainte de sécurité pour éviter une collision entre deux drones aériens), les points de passage pour chaque type de robot, les points à observer dans la zone, les points de rendez-vous, les points de départ pour chaque robot et les patrouilles prédéfinies par type de robot.

Le plan initial est présenté sur la figure 6.9. Cette figure ne concerne que les robots réels, les robots de remplacement n'ayant pas de plan initialement, ils ne sont pas représentés.

Les objectifs de la mission sont (dans l'ordre de priorité) :

– Suivi des cibles : c'est le but prioritaire de la mission.

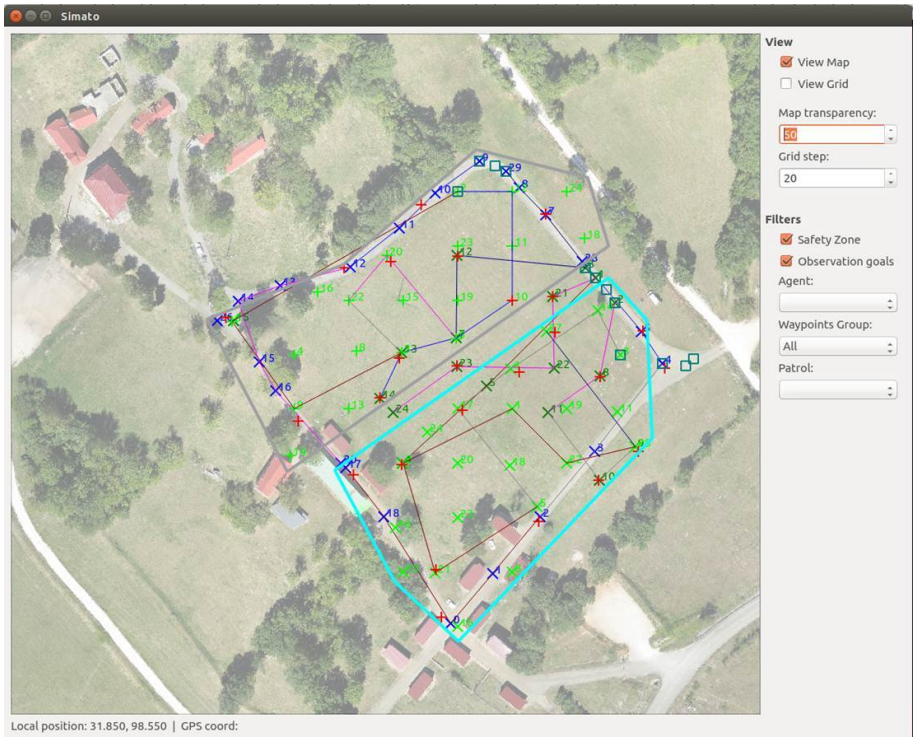


Figure 6.8 – Exemple de capture d'écran de l'interface de préparation de mission SIMATO.

– Exploration des points à observer : on s'autorise à ne pas observer un point uniquement si aucun robot ne peut plus l'observer.

– Respecter les rendez-vous : si un rendez-vous n'est plus tenable, il est abandonné. Les rendez-vous font partie de la définition de la mission, ils sont définis par l'opérateur et contiennent les deux robots qui doivent se retrouver, la date précise et le lieu de leur rendez-vous. Si une réparation échoue à cause d'un plan incohérent temporellement, METAL abandonne les rendez-vous restants de l'agent avant de réessayer de réparer le plan.

Pour garantir la priorité du suivi de cible, de nouveaux états sont introduits dans la machine à état de METAL. Dès qu'un robot détecte une cible, il passe dans l'état *Tracking* et arrête d'exécuter son plan pour suivre la cible. Les autres robots attendent les consignes de l'opérateur : d'autres robots peuvent recevoir l'instruction de suivre la cible. Une fois que l'opérateur a donné ces consignes, un robot reçoit l'ordre de continuer la mission. Il doit alors réparer le plan, comme si les robots suiveurs étaient devenus indisponibles. Un robot en train de suivre une cible peut aussi recevoir l'ordre d'arrêter le suivi, il repasse alors dans l'état *Running*. Cette machine à état est représentée sur la figure 6.10.

De plus, lors de l'exécution d'une mission, on veut pouvoir contrôler quelles communications sont possibles pour tester la robustesse de l'équipe aux pertes de communication. Pour cela, toutes les communications de la mission (concernant la synchronisation des MacroSTN, les réparations, les rendez-vous) passent par un serveur centralisé. Cela permet à l'opérateur de donner des consignes pour bloquer ou faire passer certains messages en fonction de l'émetteur, du récepteur, de la position des robots, etc. Dans toutes les missions

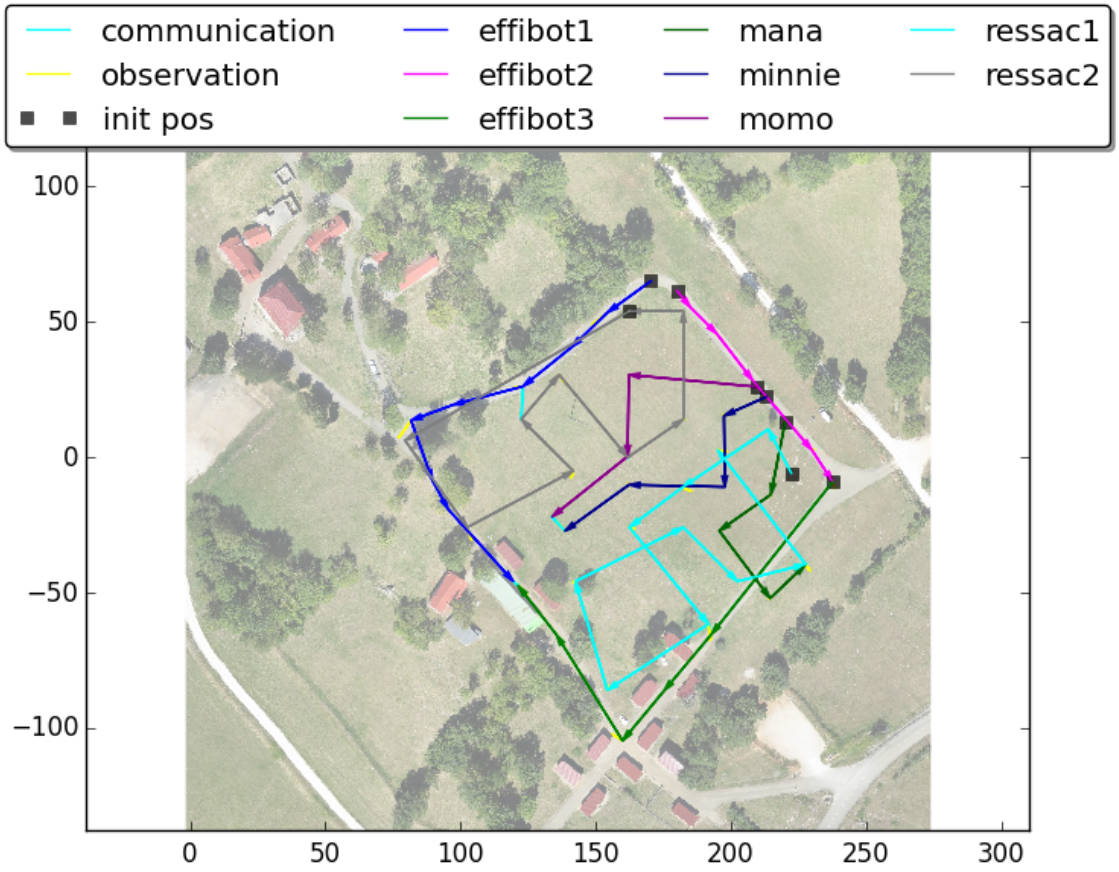


Figure 6.9 – Trajectoire des robots (hors robots de remplacement) dans le plan initial. Les points de rendez-vous sont liés par un trait cyan pour indiquer qu'une communication y aura lieu. Les points à observer sont liés par un trait jaune au point de passage auquel le robot sera pour les observer.

qui suivent, on ne s'en sert que pour isoler des robots individuels du reste de l'équipe. Ce serveur centralisé permet aussi de mesurer le nombre de messages échangés, la bande passante utilisée, etc.

Une autre adaptation de ce qui a été présenté jusqu'ici concerne les échéances. Les rendez-vous imposés dans le plan sont associés à des échéances ayant une date fixée. Le plan initial contient donc ces tâches de rendez-vous. Néanmoins, durant l'exécution, il n'est pas souhaitable que le plan soit invalidé si les deux robots sont en retard, le rendez-vous pouvant être reporté. De même, si les deux robots sont là en avance il n'est pas nécessaire d'attendre la date exacte du rendez-vous pour communiquer. Mais il est quand même souhaitable d'avoir une borne permettant à un robot de ne pas être bloqué indéfiniment en attendant un autre robot. Pour cela, METAL ignore les dates précises des échéances de communication lors de l'exécution : il retire la borne supérieure et s'autorise à l'exécuter au plus tôt. Mais dès qu'un robot est prêt à communiquer (*i.e.* quand il commence son action `com-passive`), il définit une borne supérieure pour l'action `com-active`. Cette borne est calculée comme la date au plus tôt prévue dans le plan actuel plus 30 secondes. Cela signifie que si un robot est prêt à communiquer, il impose une échéance à l'autre robot de

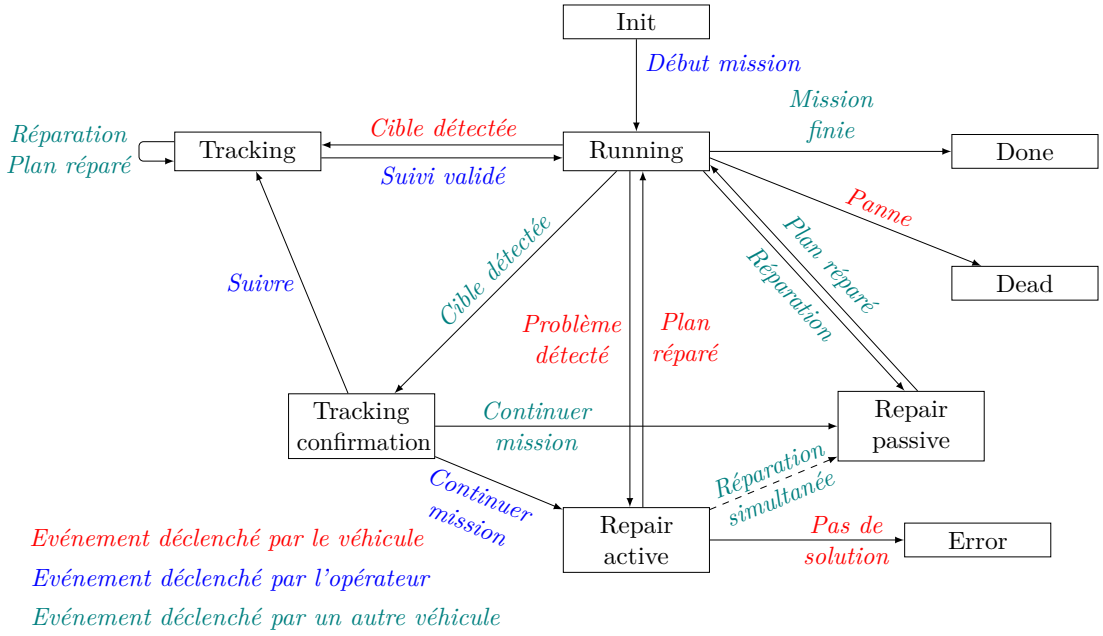


Figure 6.10 – Machine à état interne de METAL, incluant les états nécessaires au suivi de cible. Les transitions sont classées en 3 types : celles qui sont causées par le robot lui-même, celles causées par un autre robot et celles causées par l'opérateur.

ne pas prendre plus de 30 secondes de retard par rapport à ses prévisions actuelles. Ainsi le système doit toujours gérer des échéances, mais l'exécution est plus souple sur les dates des rendez-vous.

6.3 Validation expérimentale en simulation

6.3.1 Protocole expérimental

Dans un premier temps, nous avons cherché à qualifier les performances de cette architecture d'exécution de mission. Pour cela nous avons mis en place une simulation basique de l'exécution des actions.

On simule un robot qui exécute toutes les actions qu'on lui demande en attendant la durée nominale de l'action avant de confirmer son exécution. Ce robot simulé peut recevoir des consignes extérieures : allonger la durée d'une action en cours, se désactiver, remonter l'information d'une cible détectée, etc. On utilise alors ces robots simulés simplistes, sans avoir besoin de simuler un environnement réaliste, pour faire un grand nombre de simulations.

En l'absence de modèle d'exécution réaliste (par exemple combien de temps prend réellement un déplacement, comment un robot va réagir à un nouvel obstacle sur sa route, quel est l'effet du vent sur la dynamique des AAV, etc.), il n'est pas possible de réaliser des statistiques représentatives d'une exécution réelle. Mais il est possible de simuler la réaction à certains aléas dans des cas idéalisés d'exécution. C'est l'objet des simulations statistiques que nous présentons ici.

Nous avons défini un ensemble de *patrons*. Chaque patron permet de générer des

scénarios de test aléatoirement. Un scénario de test définit précisément quels aléas arrivent, à quelles dates et pour quels robots. Par exemple le patron « une cible est détectée » permet de générer un scénario où l'AGV Mana détecte une cible une minute après le début de mission et un autre où c'est l'AGV Effibot1 qui détecte la cible deux minutes et trente secondes après le début de la mission.

Pour chaque patron, on tire au hasard un certain nombre de scénarios de test. On simule une mission par scénario de test et on calcule des métriques sur tous les scénarios associés à un patron donné.

Les résultats présentés ci-dessous ont été obtenus sur un PC de bureau (Intel Xeon 4 cœurs cadencés à 2.67GHz, 6GB de RAM). Les simulations ont été réalisées en temps réel avec 3 simulations en parallèle.

Les patrons utilisés sont (où AXV désigne un robot autonome qu'il soit terrestre ou aérien) :

- **Nominal** : aucun aléa, c'est le scénario de référence
- **Un AXV devient non opérationnel** : à une date aléatoire un des robots tombe en panne. A une autre date aléatoire, après la précédente, un autre robot en est informé et répare le plan.
- **Un AXV devient non opérationnel** et **un AXV ne peut plus communiquer** : comme précédemment, mais un troisième robot est isolé du reste de l'équipe avant la perte du robot et jusqu'après la réparation.
- **Un AXV détecte une cible** : À une date aléatoire, un robot détecte une cible. Cinq secondes plus tard, un autre robot reçoit l'ordre de l'opérateur de continuer la mission. Une minute après, le robot ayant détecté la cible reçoit l'ordre d'arrêter le suivi (il retourne dans l'équipe).
- **Un AXV détecte une cible** et **un AXV ne peut plus communiquer** : comme précédemment, mais un troisième robot est isolé du reste de l'équipe avant la détection de la cible et jusqu'après la réparation.
- **Un AXV détecte une cible (répété 2 fois)** : on tire deux aléas « **Un AXV détecte une cible** », impliquant des couples de robots différents.
- **Un AXV est en retard** : On tire au hasard une date et un robot, l'action qu'effectue ce robot à cette date se voit retardée de 45 secondes.
- **Un AXV est en retard** et **un AXV ne peut plus communiquer** : comme précédemment, mais un deuxième robot est isolé du reste de l'équipe avant le retard jusqu'à une date aléatoire ultérieure.
- **Deux aléas sont générés** : on tire au hasard deux aléas parmi tous ceux présentés précédemment, séparés d'au moins 20 secondes.
- **Cinq aléas sont générés** : on tire au hasard cinq aléas parmi tous ceux présentés précédemment, séparés d'au moins 20 secondes. Pour garantir qu'il y ait suffisamment de robots disponibles, on s'autorise à choisir des robots de remplacement lors de la génération des aléas.

Pour chaque patron, on s'intéresse à l'évolution des métriques suivantes :

– La durée de la mission : la fin de la mission étant définie comme le dernier instant où un robot exécute la fin globale de son plan ;

– Le nombre de points observés ;

– Le nombre de rendez-vous effectués ;

– Le nombre total de messages envoyés ;

– La durée moyenne d'une réparation : mesurée entre la date d'envoi du message demandant une réparation par le robot réparateur et la date d'envoi du plan réparé.

6.3.2 Résultats expérimentaux

Une première analyse montre que sur les 300 scénarios simulés, toutes les missions se sont soldés par des succès : tous les superviseurs ont terminé dans l'état *Done* de leur machine à état (ou *Dead* pour ceux qui ont été rendus indisponibles). Sur tous les patrons présentant des cibles, le nombre de suivis de cible est égal au nombre de cibles. Cela montre que l'objectif prioritaire de la mission est correctement réalisé.

La figure 6.11 montre l'évolution de la durée de la mission.

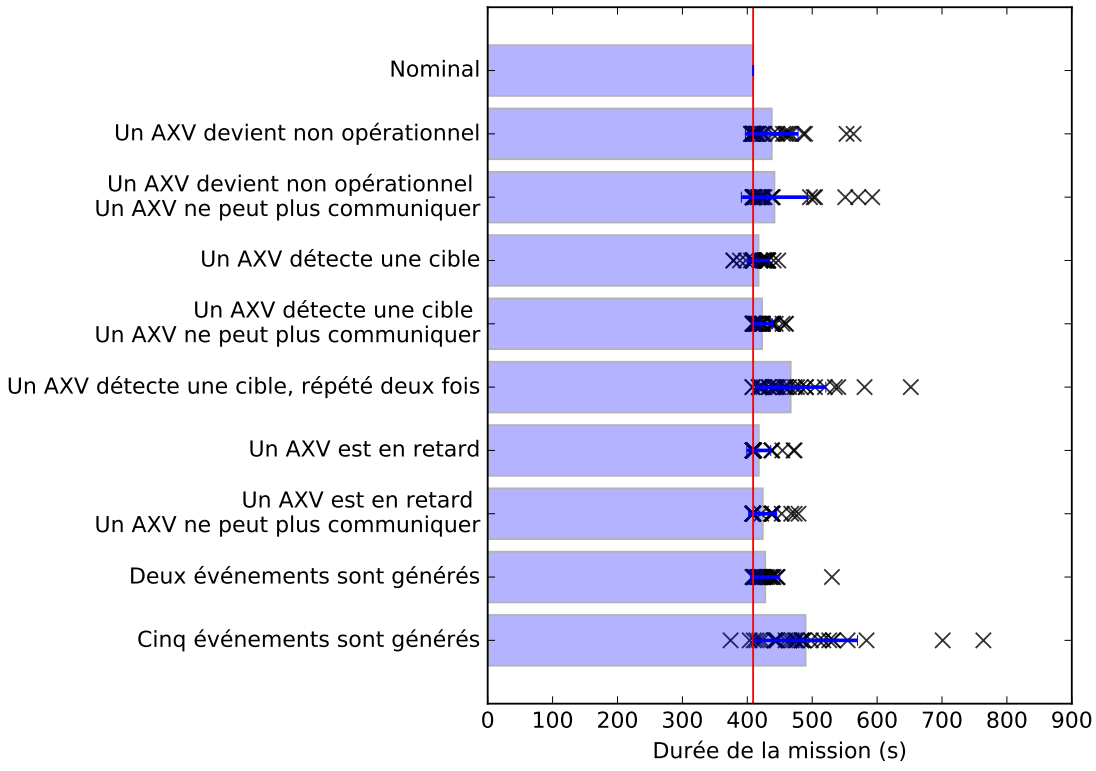


Figure 6.11 – Évolution de la durée de la mission en fonction du patron utilisé. Chaque croix représente un scénario de test. La barre horizontale représente la moyenne des 30 scénarios. La barre verticale rouge correspond à la moyenne dans le scénario nominal.

Premièrement, on voit que la durée moyenne d'une mission augmente quand un aléa est introduit. Si on introduit plusieurs aléas, la durée moyenne d'une mission augmente d'autant plus. Avec un aléa elle augmente en moyenne de 4.4%, avec 2 aléas de 9.3% et avec 5 aléas elle augmente de 19.8%.

Dans certains cas particuliers, la durée de la mission peut aussi diminuer. Ce cas peut se produire si le robot ayant le plus long plan (donc celui qui définit la durée de la mission) a un problème lors d'une patrouille après avoir observé tous les points qu'il devait observer mais avant d'avoir fini sa patrouille. La réparation va alors considérer que la patrouille a rempli tous ses objectifs, même si elle n'est pas complète. Aucune nouvelle action ne sera ajoutée et le plan sera donc raccourci. La perte de communication semble aussi avoir un impact négatif, quoique limité, sur la durée des plans. Dans tous les cas, la durée de la mission reste proche de sa moyenne dans le cas nominal : il n'y a pas d'explosion de la durée quand des aléas surviennent.

La figure 6.12 montre l'évolution du nombre de points observés lors d'une mission.

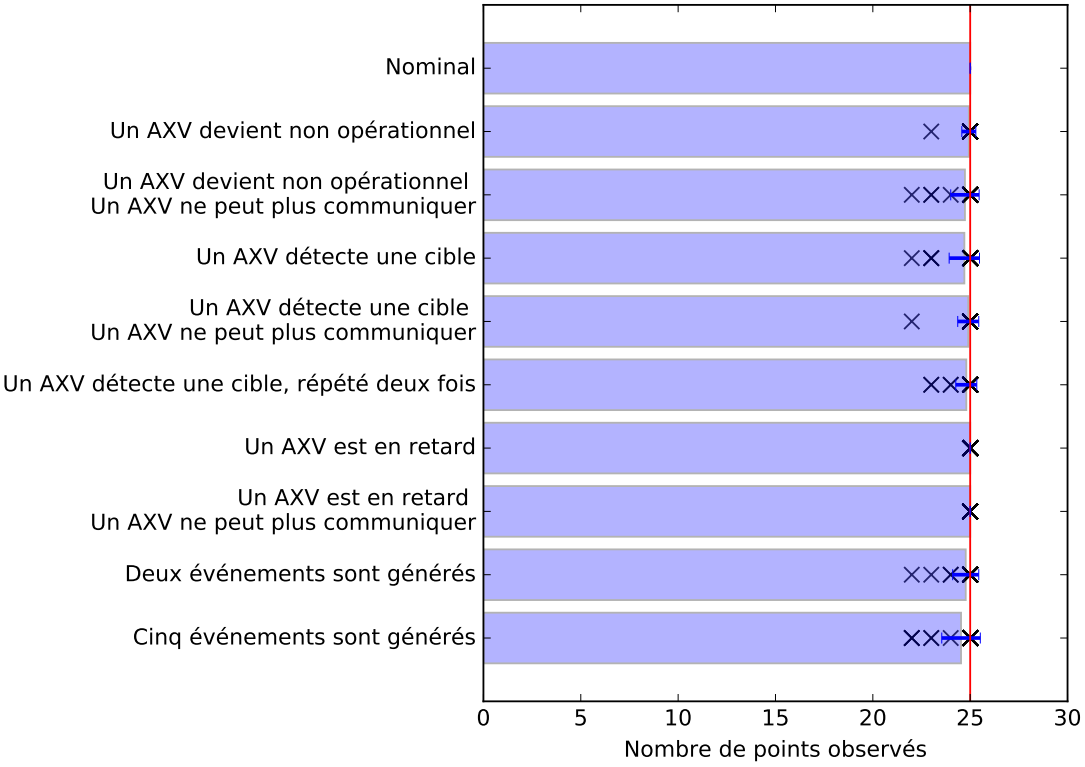


Figure 6.12 – Évolution du nombre de points observés en fonction du patron utilisé. Chaque croix représente un scénario de test. La barre horizontale représente la moyenne des 30 scénarios. La barre verticale rouge correspond à la moyenne dans le scénario nominal.

On peut voir ici l'influence des robots de remplacement. Tous les robots sauf un AAV ont au moins un robot de remplacement possible, ce qui veut dire qu'un seul robot est indispensable à l'exploration de tous les points. Si ce robot devient indisponible (s'il tombe en panne ou s'il détecte une cible), alors certains points ne sont plus explorables. S'ils n'ont pas encore été explorés, alors ils sont abandonnés par HiPOP lors d'une réparation (cf.

section 5.3.2). Il n'y a que 3 points explorables que par cet AAV, c'est donc le maximum de points à explorer qui peuvent être abandonnés. C'est ce que l'on observe sur les résultats : on ne descend jamais en dessous de 22 points observés sur les 25. Et à moins d'avoir cet AAV indisponible (ce qui est un événement peu fréquent quand les aléas sont tirés au hasard), tous les points sont observés. C'est pour cela que la moyenne du nombre de points est aussi proche de 25.

La figure 6.13 montre l'évolution du nombre de rendez-vous effectués lors d'une mission.

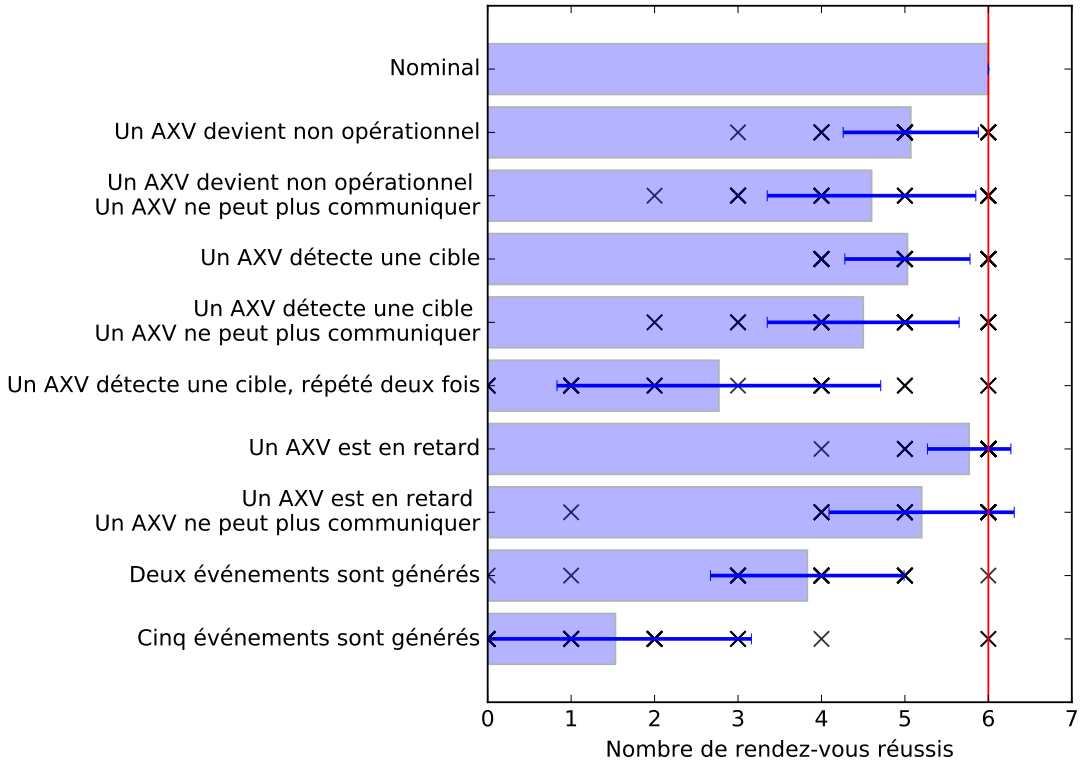


Figure 6.13 – Évolution du nombre de rendez-vous effectués en fonction du patron utilisé. Chaque croix représente un scénario de test. La barre horizontale représente la moyenne des 30 scénarios. La barre verticale rouge correspond à la moyenne dans le scénario nominal.

La mission initiale prévoit 6 rendez-vous. Il est possible d'abandonner des rendez-vous mais pas d'en ajouter, c'est ce qui explique que le nombre de rendez-vous est toujours inférieur à 6. On note que quand la communication est interrompue, le nombre de rendez-vous effectués diminue (ce qui est attendu, les communications peuvent aussi être coupées aux points de rendez-vous). Plus on a d'aléas, plus le nombre de rendez-vous irréalisables augmente. On voit aussi qu'un retard permet dans certains cas d'empêcher un rendez-vous : il faut que le retard se déclenche quand l'autre robot est déjà en train d'attendre. Cela limite le nombre de rendez-vous abandonnés en présence de retards.

La figure 6.14 montre l'évolution du temps moyen de chaque réparation.

Cette durée regroupe le temps nécessaire pour faire la fusion des plans locaux en un plan global et la réparation du plan global. Avant de faire la fusion des plans locaux, le

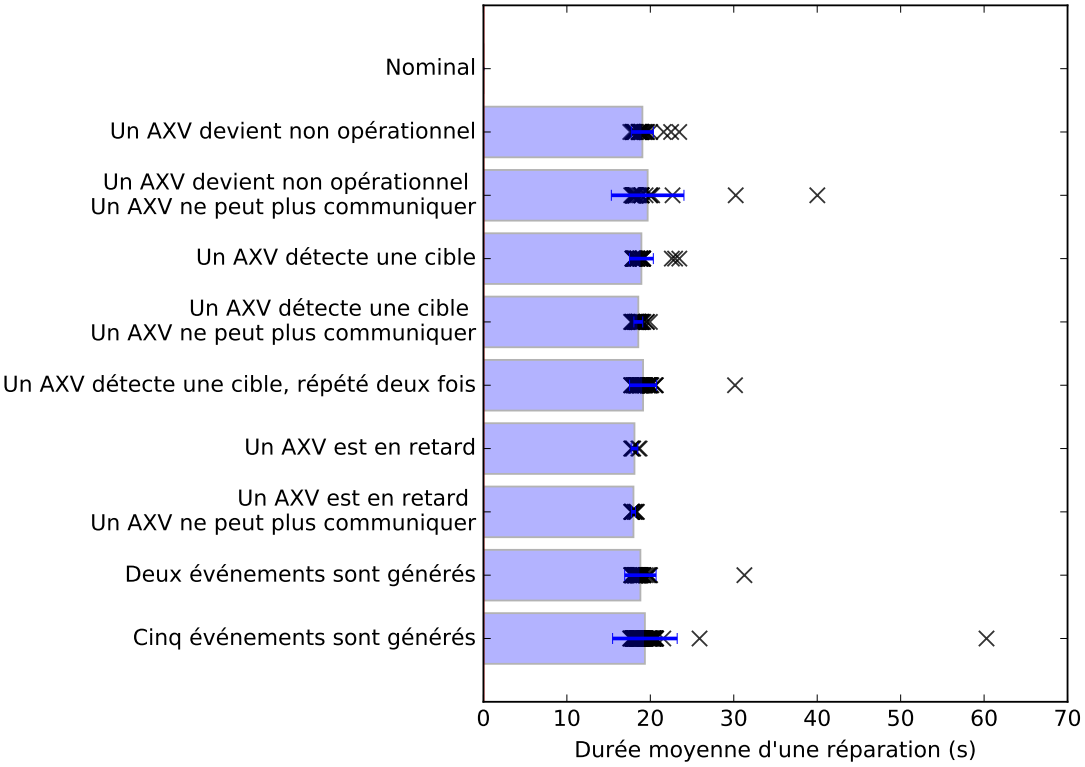


Figure 6.14 – Évolution du temps moyen de chaque réparation en fonction du patron utilisé. Chaque croix représente un scénario de test. La barre horizontale représente la moyenne des 30 scénarios. La barre verticale rouge correspond à la moyenne dans le scénario nominal.

robot réparateur contacte tous les autres robots pour récupérer leurs plans et attend 10 secondes pour avoir leurs réponses. Cette durée de 10 secondes incompressible est incluse dans cette moyenne. On note donc que dans tous les cas, le temps moyen d'une réparation est de 10 secondes environ. Seuls certains cas particuliers nécessitent plus de 20 secondes de réparation.

La figure 6.15 montre l'évolution du nombre total de messages envoyés à chaque mission.

Comme pour la durée de la mission, le nombre de messages envoyés augmente avec le nombre d'aléas ou quand un robot est isolé du reste de l'équipe. Il reste néanmoins proche de sa moyenne dans le cas nominal : il n'y a pas d'explosion du nombre de messages.

En conclusion, nous avons montré la validité de l'approche : les missions sont correctement exécutées en présence d'échéances à respecter, de pertes de communication et de divers aléas pouvant survenir. La réparation permet bien de réparer des plans en cours d'exécution et possédant des échéances (ce qui n'avait pas été montré expérimentalement au chapitre précédent sur une large gamme de réparations). Le temps de réparation moyen est limité et permet d'envisager de réaliser des missions en conditions réelles. On a aussi pu étudier l'influence de certains types d'aléas dans des conditions d'exécution idéales et montrer la robustesse de l'exécution à ces aléas.

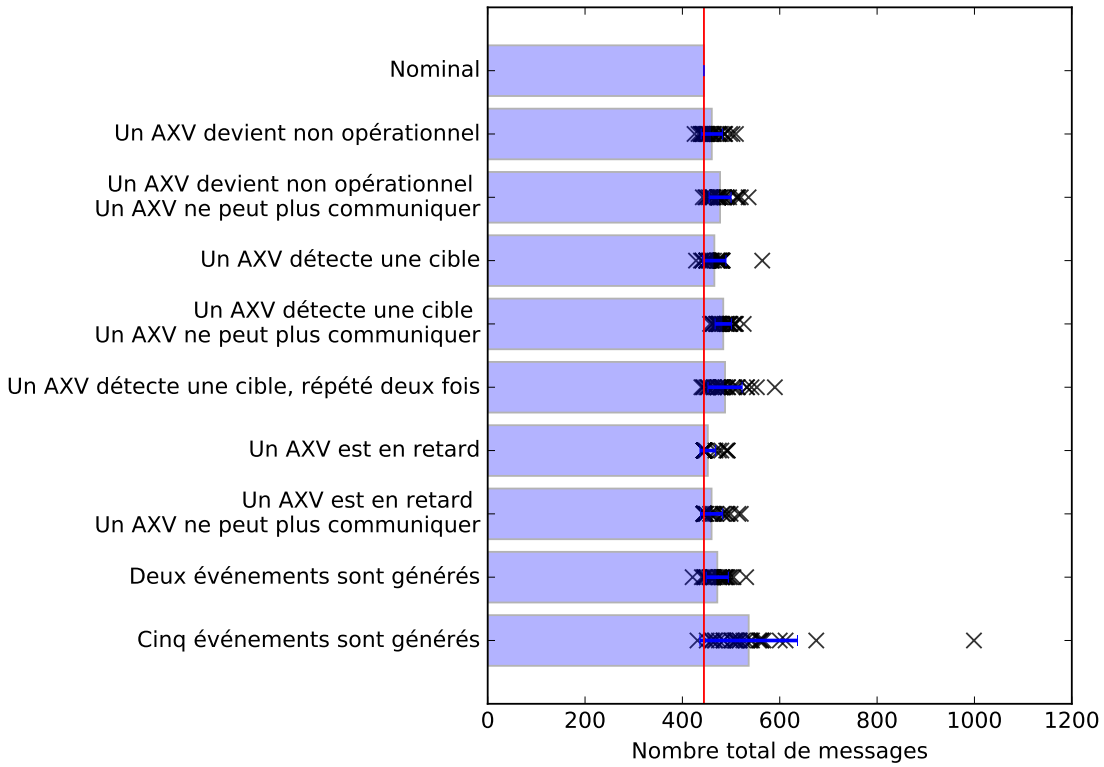


Figure 6.15 – Évolution du nombre total de messages envoyés en fonction du patron utilisé. Chaque croix représente un scénario de test. La barre horizontale représente la moyenne des 30 scénarios. La barre verticale rouge correspond à la moyenne dans le scénario nominal.

6.4 Validation expérimentale en conditions réelles

L'architecture composée d'HiPOP et de METAL a été utilisée pour les scénarios V et VI du PEA ACTION. La démonstration du scénario V a eu lieu le 18 juin 2015 au village de combat du camp militaire de Caylus. La démonstration du scénario VI a eu lieu les 19 et 20 octobre 2015 sur ce même site. Les résultats présentés ci-dessous concernent ceux obtenus lors de la démonstration du scénario VI. Au total, 9 missions ont été jouées.

A la suite d'un problème technique avant le début des démonstrations, un AGV du LAAS était indisponible. Toutes les expérimentations, sauf mentions contraires, ont donc rassemblé sept robots réels (deux AAV de l'ONERA, deux AGV du LAAS et trois AGV de la DGA) ainsi que cinq robots simulés (dont un AGV avait des actions à effectuer dans la mission initiale et quatre étaient des robots de remplacement).

De plus, au cours des démonstrations ,certaines adaptations au plan initial ont été nécessaires. Initialement, les AGV de la DGA étaient cantonnés aux chemins alors que les AGV du LAAS pouvaient se rendre dans les zones herbeuses (ils ont une meilleure garde au sol et des capteurs plus adaptés). Néanmoins les premières expérimentations ont montré que certaines trajectoires sur les chemins nécessitaient l'utilisation des AGV du LAAS alors que certaines trajectoires dans l'herbe pouvaient être réalisées par des AGV de la DGA. Ainsi, au cours de la démonstration, certains robots ont été échangés dans le plan.

Une de ces missions s'est soldée par une exécution nominale : tous les robots ont terminé leur plan nominal et tous les rendez-vous ont eu lieu. L'état du plan global exécuté est présenté sur la figure 6.16. Les trajectoires des différents robots sont présentées sur la figure 6.17.

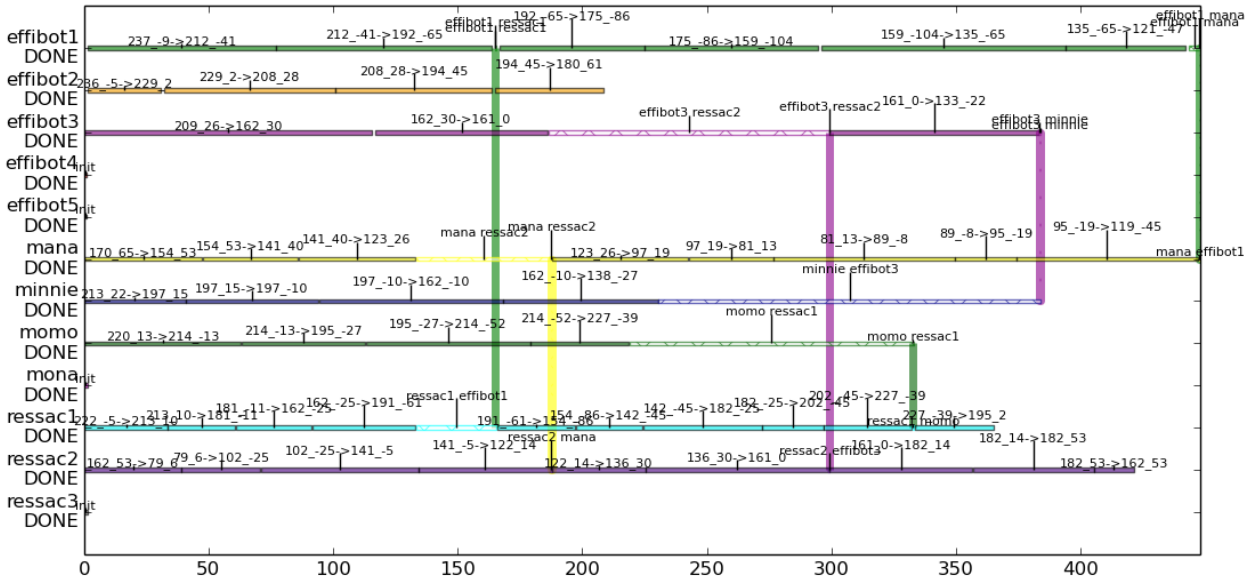


Figure 6.16 – Capture d'écran de l'interface de visualisation de l'exécution Timeline à la fin de l'exécution d'une mission nominale.

Les trajectoires représentées incluent aussi les trajectoires des robots après la fin de la mission. C'est ce qui explique que certains robots reviennent à leur point de départ (comme Mana et Effibot1 qui traversent le champ central alors que leur dernière action était un rendez-vous ensemble). Ces trajectoires sont aussi à comparer avec le plan initial (figure 6.9 page 173). On voit que les trajectoires des AAV sont très soumises aux aléas du vent : les robots rallient les points de passage mais ne le font pas forcément en ligne droite. Les robots terrestres ont quand à eux une dynamique plus prédictible.

Cette mission a aussi connu certains retards (elle a duré 450 secondes au lieu des 410 prévues). En particulier les déplacements de Mana et d'Effibot1 ont été retardés. Le rendez-vous entre ces deux robots n'aurait pas pu arriver à la date prévue dans le plan, mais comme les deux robots étaient en retard avant que l'un des deux ne commence son attente, le rendez-vous a été correctement décalé dans le plan. Cela a permis de réaliser ce rendez-vous sans déclencher de réparation.

Une mission a permis de mettre en évidence la robustesse de l'architecture à la perte de plusieurs robots. L'état du plan global exécuté est présenté sur la figure 6.18. Les trajectoires des différents robots sont présentées sur la figure 6.19.

On voit sur la figure 6.18 que deux robots ont fini la mission dans l'état *Dead* : Effibot1 et Momo. Dans cette mission, Momo avait échangé sa place avec Effibot3. C'est pour cela que les deux robots de remplacement qui ont été utilisés sont les Effibot de remplacement : Effibot4 et Effibot5.

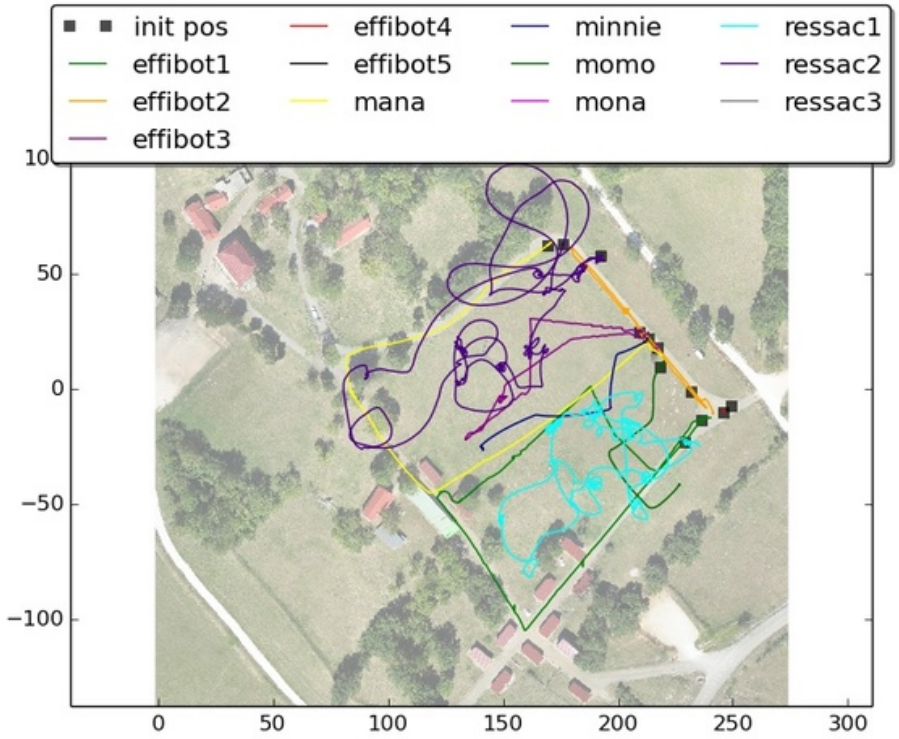


Figure 6.17 – Trajectoire des différents robots lors de l'exécution d'une mission nominale.

On remarque aussi qu'il s'est écoulé un certain temps entre la perte des robots et le début des actions de remplacement (environ 400 secondes soit environ 6 minutes). Cela est dû à un choix de l'opérateur : le reste de l'équipe n'a été notifiée que plus tard de la panne des deux robots. Cela a entraîné une attente des robots ReSSAC1 et ReSSAC2 : un rendez-vous était prévu et en l'absence de nouvelles informations ils ont attendu. Puis le rendez-vous a été abandonné : la panne de ces deux robots, même non détectée par le reste de l'équipe, n'a pas bloqué la mission. Lorsque l'équipe a été notifiée de la mort des robots, des réparations ont été déclenchées, affectant les tâches restantes aux robots de remplacement. La mission s'est ensuite terminée correctement.

Sur les trajectoires effectuées, on peut voir l'arrêt d'Effibot1 sur le chemin (aux alentours du point (220,-50)). Il est remplacé par Effibot4 (en rouge). Effibot4 ne suit pas la patrouille traditionnelle et se rend directement au point à observer (dans le village, au sud). Cela est possible car une patrouille à moitié exécuté a été abandonné, le planificateur a donc pu ajouter toutes les actions élémentaires disponibles (cf. section 5.2.2). Momo lui s'est arrêté aux alentours du point (150,50), au nord. Il a été remplacé par Effibot5 (en noir) qui a repris toute la patrouille de Momo (initialement celle d'Effibot3 dans le plan initial).

Une mission a permis de mettre en évidence le comportement de l'équipe lorsqu'une cible est détectée. L'état du plan global exécuté est présenté sur la figure 6.20. Les trajectoires des différents robots sont présentées sur la figure 6.21.

Cette mission a été arrêtée volontairement avant sa fin. Néanmoins avant cela Mana a eu le temps de détecter la cible. Mana est alors passé en suivi de cible, les autres robots ont mis leur plan en pause en attendant les consignes de l'opérateur. Une fois la consigne

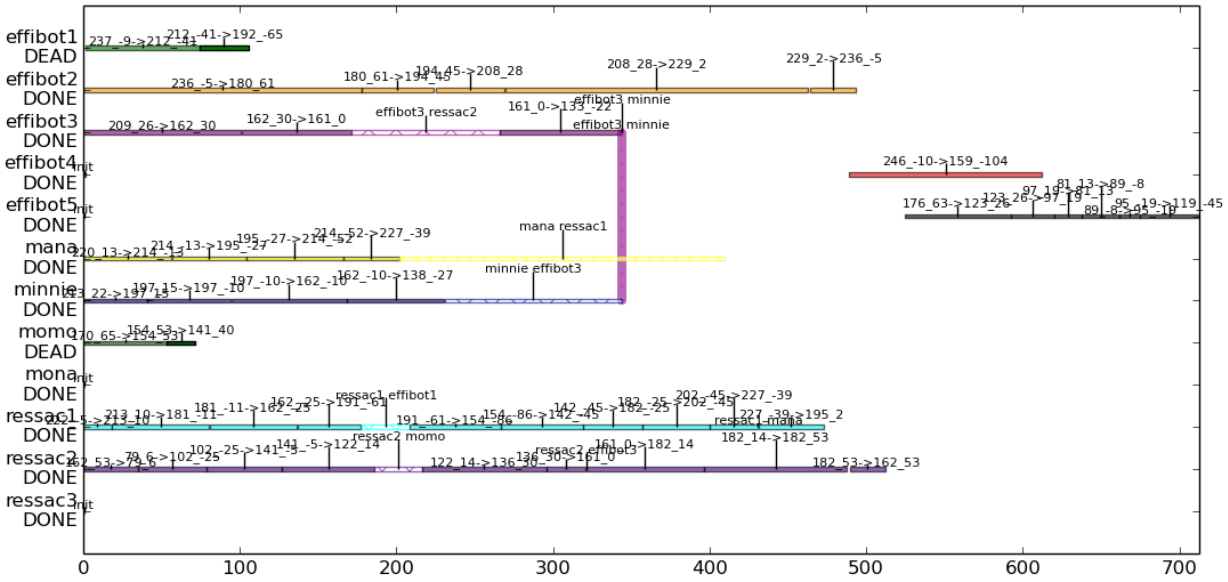


Figure 6.18 – Capture d'écran de l'interface de visualisation de l'exécution Timeline à la fin de l'exécution d'une mission avec deux robots indisponibles.

de poursuivre le plan donnée, l'exécution a repris. C'est ce qui explique le trou dans le plan de presque tous les robots avant t = 150 : aucune nouvelle action n'était lancée, seules les actions en cours d'exécution ont continué. Mana étant indisponible pendant son suivi de cible, Mona (le nom du robot de remplacement de type AGV LAAS) a donc repris ses actions restantes. ReSSAC1 étant aussi mort, ReSSAC3 a pris sa place.

Ces démonstrations ont été un succès. Les résultats de toutes les missions jouées sont disponibles à l'annexe E. Ces démonstrations ont validé expérimentalement plusieurs développements scientifiques sur la localisation et navigation autonome de robots terrestres et sur la coopération entre robots hétérogènes. En particulier elles ont permis de valider l'architecture de coopération basée sur le planificateur HiPOP et le superviseur METAL. L'équipe de robots dans son ensemble a accompli ses objectifs même en présence de retards importants lors de l'exécution de certaines actions, de robots inopérants ou de cibles présentes sur la zone.

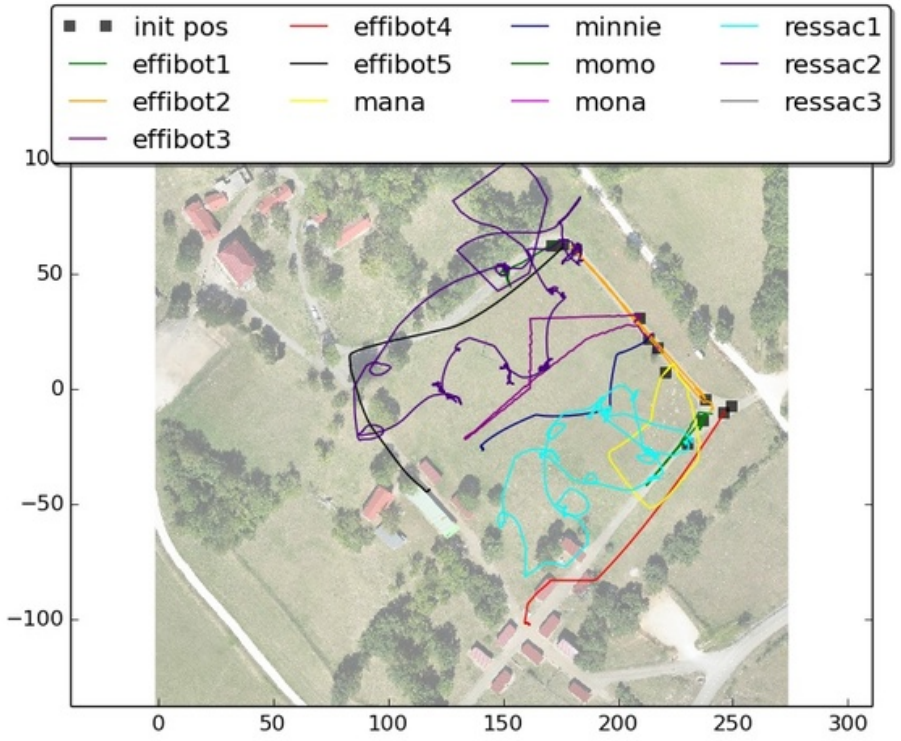


Figure 6.19 – Trajectoire des différents robots lors de l'exécution d'une mission avec deux robots indisponibles.

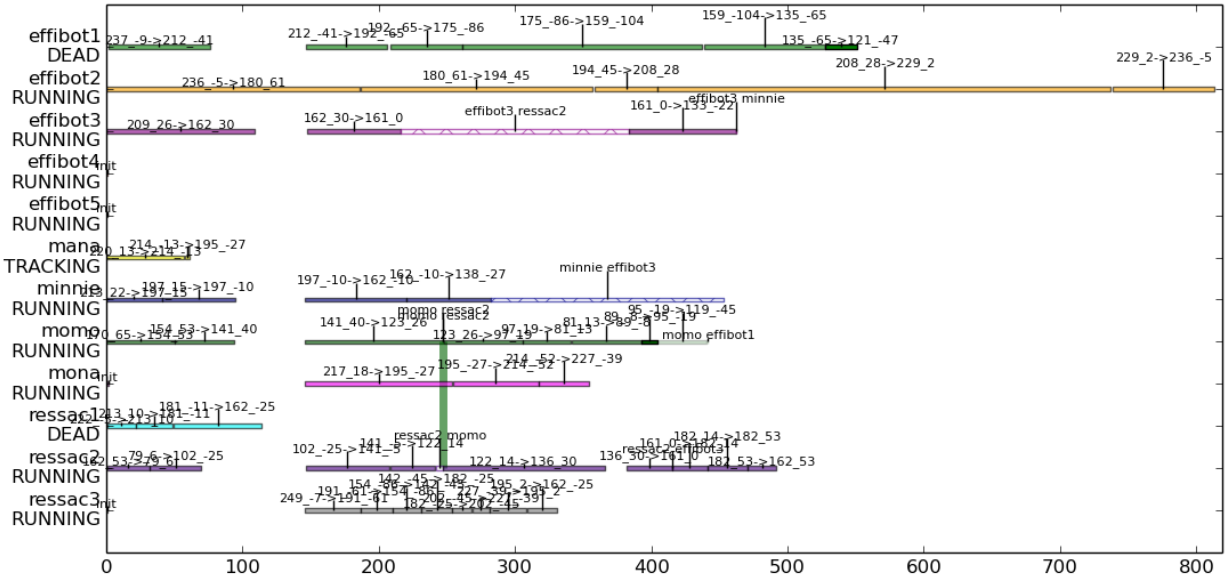


Figure 6.20 – Capture d'écran de l'interface de visualisation de l'exécution Timeline à la fin de l'exécution d'une mission où une cible est détectée.

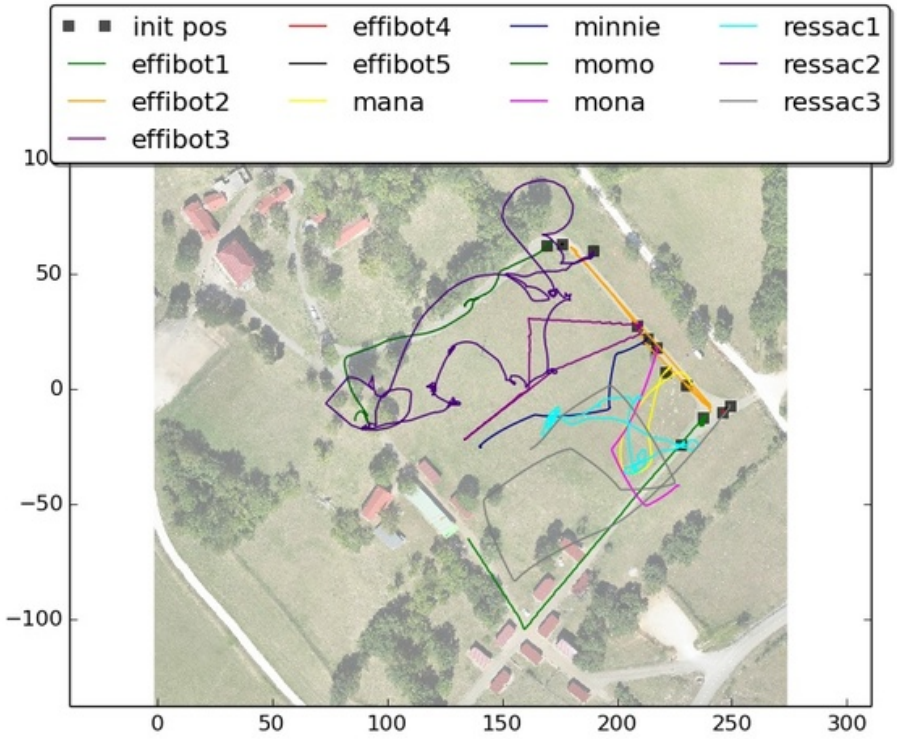


Figure 6.21 – Trajectoire des différents robots lors de l'exécution d'une mission où une cible est détectée.

Conclusion

Dans ce chapitre, nous avons montré comment superviser un plan flexible temporellement de manière distribuée et comment réaliser des missions complètes à l'aide de ce superviseur et du planificateur précédemment décrit.

Le superviseur METAL repose sur une machine à état interne et sur l'exécution d'un plan flexible temporellement. Cette flexibilité permet d'absorber la majorité des retards qui peuvent survenir. En cas de retard empêchant une échéance d'être réalisée ou d'un problème plus grave, le planificateur HiPOP est utilisé pour réparer le plan. Cela nécessite de fusionner les plans en cours d'exécution de chaque robot avant de réparer le plan global.

Ce superviseur a été adapté pour permettre l'exécution de missions dans le cadre du PEA ACTION. Les missions consistent à patrouiller dans une zone donnée à la recherche de cibles puis à les suivre. Les missions présentées impliquent douze robots simultanément (huit réels et quatre simulés) ce qui a nécessité d'utiliser une simulation hybride (un couplage entre la simulation et l'environnement réel).

Les résultats présentés ont montré, en simulation et lors d'exécutions réelles, la robustesse de l'architecture. En simulation nous avons pu effectuer des statistiques sur des cas d'exécution simplifiés pour étudier l'impact d'un aléa unique ou d'un ensemble d'aléas sur l'exécution du plan. On a ainsi montré qu'en présence d'aléas (et avec un modèle d'exécution des actions optimiste), la mission est exécutée au mieux. Ni la durée de la mission ni le nombre de messages échangés n'explosent et les objectifs principaux sont atteints.

Lors de missions expérimentales sur le terrain, le faible nombre de réalisations n'a

pas permis d'extraire de telles statistiques. Néanmoins, les différents scénarios joués ont donné pleine satisfaction et ont mis en évidence l'exécution correcte de la mission même en présence d'aléas.

Troisième partie

Conclusion et perspectives

Conclusion générale

Ces travaux ont permis la création et la validation d'une architecture distribuée permettant la réalisation d'une mission complexe de contrôle d'une zone par une équipe de robots dans un environnement réel et sous contraintes de communication. Cette architecture est principalement composée d'un planificateur nommé HiPOP et d'un superviseur nommé METAL.

Dans un premier temps nous nous sommes intéressé au planificateur. Le rôle du planificateur est de calculer un plan permettant d'atteindre certains objectifs en respectant certaines contraintes. On trouve dans la littérature une grande variété de planificateurs basés sur des algorithmes différents.

Le planificateur HiPOP, développé dans le cadre de cette thèse, est un planificateur hybride, mélangeant les techniques de planification POP (planification temporelle) et HTN (planification hiérarchique). Il permet de résoudre des problèmes de planification en produisant un plan à la fois hiérarchique et flexible temporellement. Cette technique a été choisie pour limiter le besoin de réparer le plan en cours de mission et pour sa capacité à utiliser des informations supplémentaires en provenance de l'utilisateur (les actions hiérarchiques) pour améliorer la recherche.

Nous avons décidé d'utiliser le langage PDDL, très utilisé dans la littérature, pour représenter les problèmes de planification afin de pouvoir réutiliser des problèmes et des planificateurs existants. Une extension à ce langage a été réalisée afin de pouvoir aussi décrire des actions hiérarchiques. Ces actions hiérarchiques sont définies avec des préconditions et des effets (et peuvent donc être introduites et manipulées dans le plan comme des actions élémentaires) mais aussi avec une liste de méthodes permettant de les réaliser (chaque action devant être remplacée par une de ses méthodes pour qu'un plan soit valide). Certaines heuristiques proposées dans la littérature ont été étudiées et adaptées pour prendre en compte ces actions hiérarchiques. En particulier, l'estimation de l'effort d'un plan prend en compte le nombre d'actions abstraites non instanciées dans le plan et les défauts concernant les actions abstraites sont traités après les autres défauts.

Ce planificateur a été testé sur plusieurs domaines : quatre domaines issus des IPC (*blocks*, *gripper*, *logistics*, *satellite*) et un domaine multirobot, nommé *survivors*, créé comme exemple d'une mission d'exploration de zone par une équipe de robots. Les différentes heuristiques présentées ont donc pu être évaluées et l'apport des actions hiérarchiques sur ces domaines a pu être mesuré. De plus, HiPOP a été comparé à trois autres planificateurs temporels (VHPOP, TFD et YAHSP). Bien que les résultats soient satisfaisants dans les domaines issus des IPC, HiPOP n'a pas été capable de résoudre les problèmes du domaine *survivors*. Une analyse plus poussée du comportement de HiPOP sur ce domaine a donc été conduite.

Cette étude a mis en évidence l'importance du raisonnement géométrique. HiPOP étant un planificateur symbolique (comme tous les planificateurs PDDL), il était incapable d'utiliser les propriétés géométriques du problème.

La première étape du raisonnement a donc consisté à caractériser les symboles (appelés *fluents*) représentant les positions des robots. Les deux propriétés retenues sont le fait

qu'une seule position puisse être vraie à chaque instant et le fait que des actions de déplacement soient disponibles. Ces actions de déplacement ont la particularité de n'agir que sur la position des robots : ainsi, quelque soit le reste du plan, il est toujours possible d'insérer une action de déplacement pour faire bouger le robot.

L'unicité des positions à chaque instant permet de détecter en amont les inconsistances dans le plan. Cela permet au planificateur de détecter que deux fluents représentant la position du même robot ne peuvent pas arriver en même temps. De plus, lorsqu'une position particulière d'un robot est nécessaire, il n'est possible d'utiliser que la dernière position connue du robot, les positions plus anciennes peuvent donc être ignorées.

La présence des actions de déplacement permet d'aller plus loin dans l'estimation de la durée nécessaire pour qu'un robot atteigne une position donnée. En effet, les actions de déplacement sont les moyens les plus courts de faire changer de position un robot. Il est donc possible de connaître précisément la durée minimale qui doit séparer deux positions différentes du même robot. De plus, comme les actions de déplacement sont toujours disponibles, il n'est pas nécessaire de les introduire trop tôt dans le plan. En les introduisant plus tard dans le plan, le planificateur a plus de latitude pour réordonner les actions dans le plan tout en étant sûr de pouvoir trouver une solution.

L'impact du raisonnement géométrique dans HiPOP a été évalué sur les deux domaines présentant des fluents de position détectés par HiPOP : *satellite* et *survivors*. Les résultats des benchmarks montrent que cela permet à HiPOP de résoudre tous les problèmes du domaine *survivors*. Nous avons aussi pu vérifier que les actions hiérarchiques sont toujours un facteur majeur d'amélioration du temps de calcul nécessaire pour résoudre des problèmes.

Après avoir développé un algorithme de planification, nous avons voulu l'adapter pour lui permettre de réparer des plans en cours d'exécution. Le but de la réparation est le même que celui de la planification (on cherche à trouver un plan permettant de résoudre un problème donné), mais en partant d'un plan initial. On cherche alors à modifier ce plan en gardant les actions passées et les actions futures encore pertinentes pour l'adapter au problème à résoudre.

Parmi les différentes techniques proposées dans la littérature, nous avons décidé d'utiliser une technique itérative. On commence par retirer certains éléments du plan et on essaye de trouver une solution uniquement en ajoutant des éléments à ce plan (appelé le point de départ de la recherche). Si aucune solution n'est trouvée, on retire alors encore plus d'éléments et on réessaye. L'avantage de cette technique est de pouvoir réutiliser une grande partie du travail réalisé sur la planification : l'ajout d'éléments est fait directement par l'algorithme de planification. Nous avons aussi formalisé la description d'un plan hybride permettant de réutiliser en plan initial un plan produit précédemment par HiPOP.

À partir du plan initial, nous avons donc proposé un processus itératif permettant de générer le plan utilisé comme point de départ de la recherche (*i.e.* de l'ajout d'éléments). Ce processus prend en compte les échéances qui ne sont pas respectées et les actions obsolètes du plan (*i.e.* celles qui ne sont plus disponibles ou qui ne servent pas à établir un but du problème). À chaque fois qu'une recherche échoue, on retire les actions qui étaient causalement liées à une des actions retirées pour générer le nouveau point de départ de la recherche. Les actions déjà exécutées ne sont pas retirées du plan. Les actions hiérarchiques sont utilisées pour garantir que si une action est retirée, toutes les actions participant au même but global sont retirées (*i.e.* quand on retire une action fille d'une action abstraite, on retire toutes les actions filles en même temps). Cela permet de garder un plan qui suit au mieux l'intention de l'utilisateur et de ne pas laisser dans les plans des actions inutiles.

L'interaction des actions hiérarchiques avec la réparation a aussi nécessité la définition explicite de fluents de bas niveau, à résoudre après tous les autres défauts.

Pour permettre une réparation distribuée du plan lors d'une mission où les communications ne sont pas garanties, certaines améliorations ont été nécessaires. En particulier les actions ont été assignées explicitement aux robots ce qui permet à HiPOP de ne modifier le plan que pour les robots qui sont à portée de communication du robot réparateur. De plus, afin de ne pas abandonner la mission si un de ses objectifs n'est plus réalisable, une nouvelle modification a été nécessaire. Lors de la recherche, si aucune action ne permet d'accomplir un but du problème alors ce but est abandonné.

Des exemples de réparations ont été présentés pour mettre en évidence le comportement de HiPOP dans deux cas de figure donnés : une réparation suite à un retard empêchant une échéance d'être respectée et une réparation suite à la casse d'un robot.

Des évaluations ont été conduites pour évaluer la capacité de HiPOP à réparer des plans. Ces tests ont porté sur le domaine *survivors* et ont mis en évidence l'intérêt de la réparation. En effet nous avons comparé les configurations de HiPOP réparant un plan à des configurations qui replanifiaient (*i.e.* auxquelles aucun plan initial n'est donné). Les configurations réparant ont résolu les problèmes plus rapidement et en changeant peu les plans (par rapport au plan initial). Nous avons aussi montré que l'utilisation des actions hiérarchiques lors de la réparation apportait un gain en temps de calcul. L'information sémantique contenue dans les actions hiérarchiques (le « pourquoi » de l'utilisation de l'action) permet de produire des plans plus pertinents pour la recherche.

Pour utiliser l'algorithme de planification et de réparation développé dans une architecture distribuée sur une équipe de robots, il a été nécessaire de pouvoir exécuter un plan flexible temporellement. C'est le rôle de METAL, le superviseur développé dans le cadre de cette thèse.

La supervision d'un plan flexible temporellement repose sur la distinction entre les événements contrôlables par le superviseur (comme le lancement d'une action) et les événements incontrôlables (comme la fin d'une action). Le superviseur a pour rôle de déclencher les instants qu'il contrôle en respectant les contraintes du plan et est informé lorsqu'un instant incontrôlable survient. Si le plan devient invalide, le superviseur fait appel à HiPOP pour réparer le plan. Des adaptations ont été faites pour permettre de distribuer efficacement l'exécution du plan sur les robots, chaque robot exécutant son plan local (constitué de ses propres actions). En particulier l'utilisation d'un MacroSTN a permis à chaque robot de se concentrer sur les contraintes temporelles qui le concernent directement. La réparation du plan, si elle implique plusieurs robots, nécessite le calcul du plan global. Les robots doivent alors s'échanger leurs plans locaux pour que le robot réparateur les fusionne pour reconstituer le plan global. Afin de faciliter l'exécution du plan, METAL utilise une machine à état interne pour représenter l'état de chaque robot. Des interfaces graphiques spécifiques ont été réalisées pour suivre l'évolution de l'exécution du plan.

Ce travail a été intégré dans le Programme d'Études Amont ACTION. Le PEA ACTION met en œuvre 8 robots réels (2 drones aériens de l'ONERA, 3 robots terrestres du LAAS et 3 robots terrestres de la DGA) et 4 robots simulés. Son but est de développer des architectures embarquées permettant de réaliser des missions de contrôle de zone dans un environnement réel. La mission à réaliser a été modélisée en PDDL et une interface graphique spécifique facilitant la préparation de la mission a été crée. La machine à état de METAL a ainsi été modifiée pour prendre en compte l'objectif prioritaire du projet : le suivi des cibles détectées.

Avant de réaliser des missions en conditions réelles, nous avons réalisé une validation expérimentale en simulation uniquement. Nous avons défini des types d'événements qui pouvaient se présenter (une cible détectée, un robot qui tombe en panne, les communications qui sont coupées et une action qui prend du retard) et nous avons généré 270 scénarios contenant un ou plusieurs de ces événements. Chaque scénario a été simulé et le comportement de l'équipe a été analysé. Les résultats ont montré que la mission finissait à chaque fois en remplissant ses objectifs prioritaires : toutes les cibles étaient suivies et tous les points à observer possibles étaient observés par un robot. L'augmentation de la durée de la mission n'a pas dépassé les 5% en moyenne quand un seul événement perturbateur était introduit. L'architecture est donc robuste (en simulation) à un certain nombre d'événements perturbant le plan courant.

Cette architecture a donc été utilisée lors des démonstrations des scénarios V et VI du PEA ACTION. Ces démonstrations se sont faites en simulation hybride puisque la majorité des robots étaient réels et évoluaient dans un environnement extérieur et que des robots simulés complétaient l'équipe. Trois missions ont été analysées plus en détail dans ce manuscrit : une mission nominale, une mission où deux robots deviennent indisponibles et une mission où une cible est détectée. Ces démonstrations ont permis de valider expérimentalement l'architecture composée de HiPOP et de METAL dans un scénario représentatif d'une mission réelle en extérieur. Elles ont montré la robustesse de l'architecture dans des conditions expérimentales réelles et face à un grand nombre d'aléas.

Perspectives

Ces travaux se sont concentrés sur la réalisation d'une architecture permettant à des robots autonomes de collaborer au sein d'une équipe. Cette architecture intègre un planificateur (HiPOP) et un superviseur (METAL). Nous allons donc nous intéresser aux limites de ces deux composantes mais aussi aux missions qui ont été exécutées dans le cadre de ces travaux.

Amélioration des performances actuelles

Planification

Les performances de HiPOP ont été suffisantes pour résoudre des problèmes issus de plusieurs benchmarks et pour permettre de réaliser les missions multirobots du PEA ACTION. Néanmoins, ses performances pourraient être améliorées. On s'intéressera ici aux points suivants :

- L'amélioration de la vitesse de résolution en utilisant de nouvelles heuristiques.
- La détection plus générale des fluents de position pour pouvoir appliquer le raisonnement géométrique dans plus de cas.
- La levée de l'hypothèse d'existence des actions de déplacement pour tout couple de points pour pouvoir appliquer le raisonnement géométrique dans plus de cas.
- Une étude de l'impact des différents choix de modélisation des actions hiérarchiques sur les performances de la planification.
- Une étude de l'impact des conditions initiales sur les performances de l'architecture sur les scénarios du PEA ACTION.
- L'utilisation d'un autre planificateur pour générer les données d'entrée utilisées par HiPOP.
- La détection automatique des domaines hiérarchiquement bien formés.

Heuristiques. D'autres heuristiques proposées dans la littérature pourraient être étudiées. On peut citer l'utilisation des landmarks, certains travaux portent par exemple sur leur utilisation lors d'une planification hiérarchique [Elkawkagy et Bercher 2012]. On pourrait aussi s'intéresser à l'utilisation d'heuristiques pour remplacer h^{add} comme h^{max} ou h^2 (cf. définition 24 page 32).

Utilisation de variables d'état. Les familles de fluents candidates pour être une famille de fluents de position sont supposées avoir une forme donnée. Quand le domaine n'a pas cette forme (comme dans *logistics*), les fluents de position ne sont pas correctement détectés par HiPOP. Dans [Helmert 2009], une technique a été proposée pour transformer un

domaine PDDL en un domaine instancié utilisant des variables d'état. Cette représentation utilise un ensemble de variables prenant leurs valeurs dans des ensembles donnés : ces ensembles sont donc de bons candidats pour être des familles de fluents de position. Même sans utiliser cette transformation automatique, utiliser une description des entrées basée sur des variables d'état plutôt que sur PDDL permettrait d'avoir plus de familles candidates. Cela permettrait aussi d'utiliser les techniques présentées à la section 4.2 pour toutes les familles de fluents en définissant des mutex pour des fluents autres que les positions géométriques.

Relaxation des contraintes sur les actions de déplacement. Une autre hypothèse contraignante ayant été faite concerne la présence des actions de déplacement pour tout couple de points (*i.e.* la présence d'une action de déplacement pour aller de n'importe quel point à n'importe quel autre point). Dans notre cas d'application, cela se justifie par l'autonomie des différents robots : les AGV du LAAS et les AAV de l'ONERA sont capables de faire une navigation globale et donc de réaliser ces mouvements. Les AGV de la DGA par contre n'ont pas nativement cette capacité : certains mouvements demandés peuvent échouer pour cette raison. Une solution serait de générer automatiquement ces actions de déplacement point à point comme une action abstraite. Ainsi dans une première étape tous les mouvements entre deux points seraient calculés. Ceux n'étant pas possibles directement par une action élémentaire seraient réalisés par une action abstraite. Cette action abstraite aurait une seule méthode composée des actions élémentaires de déplacement suivant le chemin le plus court entre ces deux points.

Influence de la description hiérarchique utilisée. Dans chaque domaine étudié, une seule description des actions hiérarchiques possible a été présentée. Certaines descriptions essayent de planifier avec plusieurs niveaux de hiérarchie, d'autres non. Une étude plus exhaustive de l'impact des différents choix de modélisation sur les performances de la planification pourrait donner des pistes sur les meilleures descriptions à utiliser. En particulier on pourrait étudier :

- L'impact des effets secondaires en fonction des heuristiques utilisées.
- L'intérêt de la planification à deux niveaux de hiérarchie. Intuitivement cette réflexion à haut niveau permet de diminuer la taille du problème lors du calcul d'un plan abstrait. Mais d'autres descriptions des actions hiérarchiques pourraient être faites pour limiter la recherche sans introduire plusieurs niveaux de hiérarchie.
- La possibilité d'introduire plus de niveaux, éventuellement explicites. Le planificateur chercherait alors en priorité un plan au plus haut niveau. Une fois ce plan obtenu, le planificateur chercherait un plan de niveau inférieur, et ainsi de suite jusqu'à obtenir un plan solution. Cela nécessiterait sans doute de définir explicitement ces niveaux : quelles actions doivent être instanciées à quels niveaux, quels fluents sont nécessaires à quels niveaux, etc.
- L'impact de l'utilisation de méthodes plus ou moins complètes : est-ce qu'il vaut mieux plusieurs méthodes complètes ou une méthode unique mais introduisant des défauts ?

Pour répondre à toutes ces questions, il est nécessaire de disposer des domaines qui s'y prêtent (pour avoir des raisonnements possibles à plusieurs niveaux de hiérarchie, pour avoir des choix plus ou moins importants à faire lors de l'instanciation des méthodes, etc.).

Influence des conditions initiales. Même sans changer de domaines pour faire varier la description des actions hiérarchiques, des études supplémentaires pourraient être conduites sur les scénarios du PEA ACTION. Les résultats de l'exécution de la mission sont sans aucun doute dépendant des paramètres initiaux. Par exemple la position initiale des robots, en particulier la position initiale des robots de remplacement, peut influer sur la durée de la mission en présence d'aléas.

La description des patrouilles est un autre facteur à prendre en compte. Le nombre de patrouilles disponibles pour explorer chaque point par exemple serait un paramètre intéressant à étudier. Dans la description utilisée pour le PEA ACTION, il est assez faible (*i.e.* chaque point peut être observé par un faible nombre de patrouilles), ce qui signifie que la recherche est facilitée : il y a peu de choix à faire. Cela rend aussi les plans exécutés plus prédictibles, ce qui a aidé à leur bon suivi par les opérateurs de sécurité. Mais cela ne permet pas forcément au planificateur de produire les plans les plus efficaces : il n'a qu'un nombre limité d'actions disponibles.

Il pourrait donc être utile d'étudier l'impact de ces paramètres d'entrée sur les résultats de la mission.

Couplage avec un autre planificateur. Dans les missions réalisées dans le cadre du PEA ACTION, les patrouilles des différents robots ont été créées par l'opérateur humain préparant la mission. Il a aussi défini les points de passage de chaque robot et les points à observer. Une partie de ce travail pourrait être automatisée.

On pourrait par exemple tirer au hasard les points à observer dans une zone donnée (en respectant certains critères de couverture) et utiliser un planificateur géométrique pour calculer automatiquement des patrouilles explorant ces points. Ou utiliser un algorithme spécifique pour choisir au mieux les points à observer étant donnée la configuration du terrain.

Cela permettrait à la fois de vérifier la robustesse de HiPOP aux variations des données d'entrée et d'introduire de l'aléatoire dans les patrouilles fournies en entrée (et donc dans les plans fournis).

Domaines hiérarchiquement bien formés. À la page 77 nous avons défini les domaines hiérarchiquement bien formés comme ceux où tout plan abstrait mène à une solution. Cette propriété permet d'oublier toutes les autres alternatives dès lors qu'un plan abstrait est trouvé, ce qui permet d'accélérer la recherche. Dans la pratique, cette propriété n'est pas garantie formellement mais elle est recherchée lors de l'écriture de tous les domaines. C'est alors à l'utilisateur de garantir cette propriété et d'en informer le planificateur. Au lieu de se reposer sur l'utilisateur, des techniques de détection automatique pourraient l'aider à garantir cette propriété en détectant si elle est vraie ou dans quel cas elle n'est pas vérifiée.

Supervision

Du point de vue de la supervision, on s'intéressera aux points suivants :

- L'utilisation de STNU pour mieux gérer les incertitudes sur les durées des actions.
- La limitation de la bande passante utilisée pour la coopération au sein de l'équipe.
- La mise en place d'une réparation plus locale pour limiter la réparation aux robots proches du robot réparateur.

– La réalisation de missions avec des équipes explicites.

STNU La gestion des contraintes temporelles par le planificateur et par le superviseur à bord de chaque robot est faite par l'utilisation d'un STN (Simple Temporal Network). Ce STN est dit cohérent s'il existe une réalisation possible de ce STN, *i.e.* s'il est possible de choisir une date précise pour chaque instant tel que toutes les contraintes soient satisfaites. En pratique, cette propriété est optimiste : le superviseur n'est pas responsable de l'exécution de tous les instants, uniquement des instant contrôlables. Il ne peut donc pas choisir une date pour chaque instant.

Pour gérer cela dans un STN, un autre outil a été proposé : le STNU (Simple Temporal Network with Uncertaincity) [Morris, Muscettola et T. Vidal 2001]. Cet outil fait la différence entre points contrôlables et incontrôlables et permet de garantir que quelle que soit la date d'exécution des points incontrôlables dans des bornes données, l'ensemble des contraintes temporelles seront bien satisfaites.

Une amélioration potentielle serait alors d'utiliser un STNU lors de la planification et lors de la supervision. Les algorithmes de vérification des STNU étant plus coûteux que ceux des STN classiques, cela impactera forcément les performances de l'algorithme. Cela nécessiterait aussi un modèle plus complexe de l'exécution des actions, avec une borne haute et une borne basse. Afin de ne pas être trop conservateur avec des bornes hautes très pessimistes, il faudrait aussi pouvoir réagir dans les cas exceptionnels où les actions prennent plus de temps que leur borne haute. De plus, aucune étude sur le passage des MaSTN utilisés lors de la supervision aux MaSTNU n'est disponible dans la littérature.

Limitation de la bande passante. Un effort supplémentaire pourrait être réalisé pour limiter l'utilisation de la bande passante lors du déroulement de la mission. Cet objectif a déjà été pris en compte (par exemple par l'utilisation de MacroSTN) mais le protocole d'envoi de message utilisé pourrait encore être amélioré. En particulier les messages ne sont pas acquittés : toutes les informations nécessaires sont ré-envoyés à chaque fois pour être robuste à une perte de la communication. L'utilisation d'un mécanisme d'acquittement des messages pourrait réduire la bande passante nécessaire en supprimant les répétitions inutiles.

Réparations locales. Pour fluidifier l'exécution de la mission, la réparation à bord du robot réparateur pourrait aussi mieux prendre en compte la proximité des autres robots. Dans ce qui a été présenté, dès qu'une réparation est lancée, l'ensemble des robots à portée de communication stoppe l'exécution des nouvelles actions (mais continue les actions en cours).

On pourrait mettre en place un mécanisme de réparation itératif à la place, qui commencerait par essayer de réparer le plan uniquement pour le robot qui détecte un aléa. Si aucune solution n'est trouvée, il contacterait les robots les plus proches et essayerait de réparer avec eux. Si cela échoue, il contacterait des robots plus éloignés jusqu'à avoir essayé de réparer avec tout le monde. Si les missions utilisent des équipes explicites, ces équipes pourraient former la base de la réparation : on essayerait de réparer d'abord au sein de l'équipe avant de contacter les autres équipes. Cela permettrait d'avoir une exécution un peu mieux distribuée avec des équipes plus indépendantes les unes des autres. Cela permettrait aussi un passage à l'échelle en terme de nombre de robots impliqués.

Missions avec des équipes explicites. Les développements présentés sur HiPOP ont tous été évalués sur le domaine multiagent *survivors*. Ce domaine a été construit pour permettre de mettre en évidence l'apport des actions hiérarchiques en permettant de raisonner avec plusieurs niveaux de hiérarchie. Les démonstrations n'ont par contre pas utilisé ce principe et ont été décrites à l'aide de patrouilles définies pour chaque robot individuellement.

Il serait alors intéressant de vérifier que l'exécution permet de gérer ce type de plan efficacement. En particulier s'il était couplé avec des réparations locales, cela permettrait de passer plus facilement à l'échelle d'un terrain plus grand avec plus de robots et des équipes explicites.

Finir la mission au mieux

L'objectif principal des démonstrations faites était de finir la mission sans erreur, *i.e.* chaque robot ayant fini son plan. C'est le principe qui a guidé l'élaboration des différents logiciels développés : on ne voulait pas abandonner la mission de tous les robots en cas d'aléa sur un robot. Cette robustesse recherchée a néanmoins conduit à moins prendre en compte les critères d'évaluation lors de la mission. En particulier, le plan n'est jamais réparé pour l'améliorer quand cela est possible : il n'est réparé que pour faire face à un aléa empêchant son exécution. C'est pourquoi une évolution potentielle serait de changer cet objectif de « finir la mission à tout prix » en « finir la mission au mieux ». Ce nouvel objectif aurait plusieurs conséquences :

- Le fait d'essayer de réparer en cas d'aléa positif (par exemple une action qui se termine plus tôt ou un nouveau robot qui devient disponible).
- La nécessité d'avoir un algorithme de réparation capable de réparer un plan valide pour l'améliorer.

Réparation en cas d'aléa positif. Pour le moment une réparation n'est lancée que dans le cas où un problème empêchant l'exécution du plan est détecté. Cela peut conduire à des comportements inattendus. Par exemple dans les missions réalisées, certains points ne sont observables que par un robot. Si ce robot devient indisponible, le planificateur produit un plan qui n'observe pas ces points. Mais si le robot redevient disponible (par exemple s'il était en train de suivre une cible et que son suivi se termine), il ne reprend pas l'exploration des points qui ont été abandonnés. Si par la suite une réparation a lieu (pour n'importe quelle autre raison), le planificateur va lui assigner des actions (car les objectifs sont juste abandonnés temporairement), mais il faut qu'un autre aléa soit déclenché. Ce comportement est dû au fait qu'une réparation n'est pas entreprise en cas d'aléa positif.

Néanmoins lancer une réparation lors d'un événement positif peut aussi être préjudiciable au plan : si aucune solution meilleure n'est disponible, tous les robots ont mis leur plan en pause pour rien et donc la mission s'allonge. C'est pour cela que ce choix n'a pas été fait, mais il pourrait redevenir pertinent si les réparations étaient locales par exemple. On pourrait alors avoir uniquement le robot redevenant disponible qui réparerait et qui reprendrait alors l'exploration des points qui auraient été mis de coté pendant son absence.

Réparation pour améliorer le plan et réparation *anytime*. Même si on réparait lors d'un aléa positif, la réparation d'HiPOP va chercher à trouver un plan le plus proche

possible du plan initial. Cela signifie par exemple que si un robot est très chargé parce que tous les robots équivalents étaient indisponibles, lorsqu'un robot redevient disponible le plan trouvé par HiPOP ne sera pas différent du plan initial. En effet le plan initial est toujours solution du problème : HiPOP ne le modifiera pas. Donc l'ajout d'un robot qui pourrait aider un robot chargé ne conduit pas à une nouvelle répartition des tâches. Si on veut résoudre la mission au mieux, il est nécessaire de moins se concentrer sur la stabilité du plan lors de la réparation et plus sur l'optimisation des critères comme la durée du plan.

Cela induirait aussi un temps de recherche plus long pour HiPOP en cours de mission : il serait sans doute nécessaire de trouver le bon compromis entre le temps de calcul supplémentaire nécessaire et le temps d'exécution économisé ainsi.

Pour aller plus loin dans cette optique, la planification *anytime* semble être la solution à privilégier. Cela consiste à chercher le meilleur plan possible tout en gardant à tout moment la possibilité de renvoyer le meilleur plan trouvé jusqu'ici. On peut ainsi arrêter la recherche à tout moment (à condition qu'un plan ait été trouvé) pour reprendre l'exécution. Plus on attend et plus le plan trouvé sera optimisé.

Les premiers essais de planification/réparation en mode anytime dans HiPOP n'ont pas été concluants. Nous avons tenté de continuer la recherche de HiPOP même lorsqu'un plan valide était trouvé en gardant en mémoire à tout moment le meilleur plan trouvé. Malheureusement, même en augmentant le temps de recherche, le plan trouvé n'était pas significativement amélioré.

Cela provient du fait que lors de la recherche, une fois un plan solution trouvé, les autres plans qui seront choisis par l'heuristique des plans seront proches de la solution trouvé initialement. L'heuristique a été conçue pour trouver rapidement un plan et donc pour étendre en priorité les plans proches d'un plan valide. De plus lors de la réparation le meilleur plan accessible à la recherche dépend des actions qui ont été retirées du plan initial. Pour pouvoir trouver un plan meilleur, il pourrait alors être utile de retirer plus d'éléments du plan initial pour vérifier si le plan trouvé n'est pas meilleur.

Tous ces arguments militent pour une utilisation de la recherche locale (cf. section 2.2.1). Cette solution a été écartée dans le chapitre 5 au profit d'une solution plus simple à mettre en œuvre : faire tous les retraits en une seule fois. Néanmoins la recherche locale permettrait sans doute de mieux réparer les plans et surtout de mieux implémenter un mode de fonctionnement *anytime* dans HiPOP. Cela permettrait de modifier localement le plan solution actuel par exemple pour essayer de l'améliorer sans avoir à refaire toute la recherche pour retirer un nouvel élément. Cette modification serait majeure dans HiPOP : toutes les heuristiques devraient être revues pour prendre en compte la possibilité de retrait d'un élément et pour empêcher les boucles lors de la recherche, les structures de données devraient être revues, les optimisations qui consistent à oublier les plans alternatifs lorsqu'un plan abstrait est trouvé devrait être modifiées, etc.

Réalisation d'autres missions

Les démonstrations présentées dans ces travaux ainsi que les évaluations statistiques ont toutes étaient faites sur une mission seulement (même si certaines simulations et démonstrations ont été faites sur une mission similaire impliquant moins de robots). Les résultats présentés sont suffisamment nombreux pour être représentatifs de cette mission mais de nouvelles expérimentations sont nécessaires pour valider cette approche dans de nouveaux cas. En particulier, on s'intéresse aux cas suivants :

– La possibilité de décrire des contraintes plus riches dans la description du problème.

– La possibilité pour le superviseur d'être informé régulièrement de l'état d'avancement d'une action ou de pouvoir l'arrêter pour permettre de mettre en place de nouvelles stratégies d'exécution.

– La formalisation d'une interface générique pour permettre à des robots différents de pouvoir collaborer sans avoir à embarquer une couche logicielle particulière.

– La possibilité d'adapter l'architecture actuelle pour réaliser des missions présentant d'autres objectifs.

Description des problèmes plus riches. Pour enrichir l'ensemble des problèmes qui peuvent être résolus par HiPOP, on pourrait ajouter de nouvelles contraintes sur le plan produit :

– Des contraintes géométriques sur les chemins empruntés (pour empêcher un robot de faire un demi-tour si sa dynamique l'interdit par exemple) ;
– Des objectifs récurrents (pour imposer une communication régulière entre plusieurs robots quelle que soit la durée de la mission par exemple) ;
– Des invariants à respecter dans la plan (pour garantir un graphe de communication connexe pendant toute l'exécution du plan par exemple) ;

Une telle modification nécessiterait de changer les descriptions des entrées pour pouvoir décrire ces objectifs (soit en utilisant une extension de PDDL soit en utilisant un autre langage comme ANML) ainsi que de modifier les heuristiques pour guider la recherche.

PDDL2.1 [Fox et Long 2003] par exemple permet de définir des quantités numériques et leurs évolutions. Cela permettrait de représenter le niveau de batterie restant d'un véhicule par exemple. PDDL3 [Gerevini et Long 2006] permet de définir des objectifs optionnels. Le plan est alors évalué selon une métrique donné et chaque objectif optionnel améliore cette métrique. Cela permettrait de pénaliser certains cas où un robot s'éloigne trop du reste de l'équipe. Le langage ANML [D. E. Smith, Frank et Cushing 2008] permettrait aussi de décrire l'évolution de certaines ressources dans le plan.

Une fois définies, ces contraintes ou préférences doivent être prises en compte lors de la planification. Les heuristiques guidant la recherche pourraient alors éliminer au plus tôt les plans ne satisfaisant pas ces contraintes ou accorder une priorité aux plans les satisfaisant. L'estimation de h^{add} devrait aussi prendre en compte la métrique spécifique du plan plutôt que la durée de l'action.

Cela nécessiterait aussi de s'assurer que la supervision puisse garantir que si ces contraintes sont violées lors de l'exécution, une réponse adéquate sera lancée (comme une réparation).

Interfaces plus riches. Au cours du projet, nous avons intégré plusieurs types de robots différents dans l'équipe : des robots simulés, des AAV et des AGV. Nous avons choisi de garder les interfaces les plus simples possibles pour pouvoir ajouter facilement de nouveaux robots et pour limiter le nombre de problèmes dus à des interactions entre METAL et les robots. Cela s'est traduit par des informations échangées limitées : le superviseur peut uniquement lancer de nouvelles actions et reçoit les comptes rendus d'exécution lorsqu'une action termine. Il n'a pas la possibilité d'interrompre une action en cours ou d'être informé de l'état d'une action en cours (il doit attendre la fin de l'action pour être informé d'un retard par exemple).

Dans la pratique nous avons aussi limité les actions exécutées par les robots réels aux déplacements : les capteurs sont en marche durant toute la durée de la mission donc les actions d'observation sont ignorées par le robot.

Enrichir cette interface en permettant plus d'interaction entre le superviseur et le robot est donc un axe d'amélioration important. Permettre au superviseur d'arrêter une action en cours permettrait de réagir plus vite à un retard éventuel pour se concentrer sur un objectif plus important par exemple. Faire remonter plus d'informations lors de l'exécution d'une action permettrait au superviseur de mieux réagir aux aléas y compris pendant l'exécution des actions (et pas uniquement en fonction de leur compte rendu d'exécution).

Ces hypothèses ont donc facilité la collaboration entre robots utilisant des architectures individuelles différentes mais certains comportements désirés (comme l'abandon d'une tâche qui a pris trop de retard) n'ont pas pu être réalisés.

Interface générique. On pourrait aller plus loin dans la collaboration entre robots hétérogènes si le comportement attendu du superviseur était standardisé. Actuellement chaque robot faisant partie de l'équipe doit embarquer HiPOP et METAL. Pour faciliter l'interopérabilité entre robots, on pourrait formaliser le comportement attendu de chaque robot et les messages échangés entre robots. Ainsi plusieurs implémentations différentes pourraient exister et chaque robot pourrait incorporer comme il le souhaite, au sein de son architecture logicielle existante, le fonctionnement nécessaire à la collaboration. Cela permettrait une vraie interopérabilité de tous les robots, sans avoir besoin d'implanter une couche logicielle identique sur chacun comme c'est le cas actuellement. Cela permettrait aussi de plus facilement intégrer des robots téléopérés ou des véhicules pilotés dans l'équipe.

Autres types de missions. Cette architecture a été testée sur des missions de contrôle de zone dans un environnement connu a priori. Il serait intéressant de vérifier le comportement de l'architecture dans d'autres types de missions collaboratives. On peut citer par exemple :

- Des missions de contrôle de zone sur un temps plus long : la mission se termine ici lorsque tous les points observables ont été observés. Mais lors d'un vrai contrôle de zone, on veut répéter les patrouilles en continu. Il faut alors prendre en compte des critères comme la durée maximale entre deux observations d'un même point. Si la mission se prolonge sur des temps supérieurs à l'autonomie énergétique des robots, il faut aussi prendre en compte leur temps de recharge, l'évolution des conditions environnementales (comme la tombée de la nuit), etc.
- Des missions de surveillance : la différence avec le contrôle de zone est que l'on veut garantir que toutes les cibles seront trouvées. Il est alors nécessaire d'intégrer un modèle de comportement de la cible lors du calcul des patrouilles à effectuer.
- Des missions d'exploration en environnement inconnu : dans ce cas les patrouilles initiales ne sont pas disponibles, il faut calculer et remettre à jour le plan au fur et à mesure que l'environnement est découvert. On pourrait imaginer alors des mécanismes automatiques de mise à jour de patrouilles au fur et à mesure que l'environnement est découvert.
- Des missions de sauvetage : en plus de la phase d'exploration, le sauvetage implique des opérations plus complexes une fois la « cible » détectée.
- Des missions de chasse aux mines sous-marines : les objectifs sont les mêmes mais les capacités des véhicules et l'environnement sont différents. Les communications sous-marines sont par exemple bien plus limitées que celles par le WiFi utilisé.

Toutes ces missions partagent des caractéristiques communes avec le contrôle de zone réalisé dans le PEA ACTION mais avec suffisamment de différences pour que des adaptations semblent nécessaires.

De plus, l'architecture a été pensée pour être robuste à une perte de communication ponctuelle. Les tests qui ont été réalisés rétablissaient une communication parfaite après l'aléa. Réaliser les mêmes missions mais avec des communications plus limitées serait aussi un test intéressant. On pourrait imaginer d'augmenter le nombre de robots qui sont isolés du reste de l'équipe, d'isoler des sous-équipes complètes, etc. Cela nécessiterait sans doute la mise en place d'un protocole de routage pour permettre aux robots de jouer le rôle de relais entre eux afin de garder la connexité maximale.

Quatrième partie

Annexes

A

Description en PDDL de l'exemple du chapitre 1

Description du domaine

```
(define (domain action-ex)
  (:requirements :strips :typing)
  (:types robot - object
          aav agv - robot
          loc - object)

  (:predicates
    (explored ?z - loc)
    (at ?r - robot ?z - loc)
    (adjacent ?z1 ?z2 - loc)
    (aav-allowed ?z - loc)
    (agv-allowed ?z - loc)
  )

  (:action move-aav
    :parameters (?r - aav ?from ?to - loc)
    :precondition (and
        (at ?r ?from) (adjacent ?from ?to)
        (aav-allowed ?from) (aav-allowed ?to))
    :effect (and (at ?r ?to) (not (at ?r ?from)) )
  )


  (:action move-agv
    :parameters (?r - agv ?from ?to - loc)
    :precondition (and
        (at ?r ?from) (adjacent ?from ?to)
        (agv-allowed ?from) (agv-allowed ?to))
    :effect (and (at ?r ?to) (not (at ?r ?from)) )
  )

  (:action explore
    :parameters (?r - robot ?l - loc)
    :precondition (and (at ?r ?l) )
    :effect (and (explored ?l) )
  )
)
```

Description du problème

```
(define (problem action-ex-p001)
```

```
  (:domain action-ex)

  (:objects
    aav1 - aav
    agv1 - agv
    cell11 cell12 cell21 cell22 - loc
  )

  (:init
    (adjacent cell12 cell22)
    (adjacent cell22 cell12)
    (adjacent cell21 cell22)
    (adjacent cell22 cell21)

    (aav-allowed cell22)
    (aav-allowed cell21)
    (agv-allowed cell22)
    (agv-allowed cell12)

    (at aav1 cell22)
    (at agv1 cell22)
  )

  (:goal (and
    (explored cell12)
    (explored cell21)
  ))
)
```

Description du domaine en PDDL 2.1

```
(define (domain action-ex-pddl2)
  (:requirements :strips :typing :durative-actions)
  (:types robot - object
          aav agv - robot
          loc - object)

  (:predicates
    (explored ?z - loc)
    (at ?r - robot ?z - loc)
    (adjacent ?z1 ?z2 - loc)
    (aav-allowed ?z - loc)
    (agv-allowed ?z - loc)
  )

  (:functions
    (distance-aav ?from ?to - loc)
    (distance-agv ?from ?to - loc)
  )

  (:durative-action move-aav
    :parameters (?r - aav ?from ?to - loc)
    :duration (= ?duration (distance-aav ?from ?to))
    :condition (and (at start (at ?r ?from))
                    (over all (aav-allowed ?from))
                    (over all (aav-allowed ?to))
                    (over all (adjacent ?from ?to)))
    :effect (and (at end (at ?r ?to))
                 (at start (not (at ?r ?from))) )
  )
```

```
30
  (:durative-action move-agv
    :parameters (?r - agv ?from ?to - loc)
    :duration (= ?duration (distance-agv ?from ?to))
    :condition (and (at start (at ?r ?from))
35                  (over all (agv-allowed ?from))
                    (over all (agv-allowed ?to))
                    (over all (adjacent ?from ?to)))
    :effect (and (at end (at ?r ?to))
                 (at start (not (at ?r ?from))) )
40  )

  (:durative-action explore
    :parameters (?r - robot ?l - loc)
    :duration (= ?duration 1)
45  :condition (and (over all (at ?r ?l)) )
    :effect (and (at end (explored ?l)) )
  )
)
```

Description du problème en PDDL 2.1

```
(define (problem action-ex-pddl2-p001)
  (:domain action-ex-pddl2)

  (:objects
5   aav1 - aav
    agv1 - agv
    cell11 cell12 cell21 cell22 - loc
  )

10 (:init
    (adjacent cell12 cell22) (adjacent cell22 cell12)
    (adjacent cell21 cell22) (adjacent cell22 cell21)

    (= (distance-aav cell21 cell22) 1)
15  (= (distance-aav cell22 cell21) 1)
    (= (distance-agv cell12 cell22) 5)
    (= (distance-agv cell22 cell12) 5)

    (aav-allowed cell22) (aav-allowed cell21)
20  (agv-allowed cell22) (agv-allowed cell12)

    (at aav1 cell22)
    (at agv1 cell22)
  )
25
  (:goal (and
    (explored cell12)
    (explored cell21)
  ))
30 )
```

B

Description en PDDL des domaines utilisés pour les benchmarks

Description du domaine *gripper*

```
(define (domain gripper-strips)
   (:requirements :strips :durative-actions)
   (:predicates (room ?r)
                (ball ?b)
                (gripper ?g)
                (at-robby ?r)
                (at ?b ?r)
                (free ?g)
                (carry ?o ?g))

   (:durative-action move
       :parameters  (?from ?to)
       :duration (= ?duration 1)
       :condition (and  (over all (room ?from))
                        (over all (room ?to))
                        (at start (at-robby ?from))
                  )
       :effect (and  (at end (at-robby ?to))
                     (at start (not (at-robby ?from)))
               )
   )

  (:durative-action pick
       :parameters (?obj ?room ?gripper)
       :duration (= ?duration 1)
       :condition  (and  (over all (ball ?obj))
                         (over all (room ?room))
                         (over all (gripper ?gripper))
                         (at start (at ?obj ?room))
                         (over all (at-robby ?room))
                         (at start (free ?gripper))
                   )
       :effect (and (at end (carry ?obj ?gripper))
                    (at start (not (at ?obj ?room)))
                    (at start (not (free ?gripper)))
               )
   )
```

```
    (:durative-action drop
        :parameters  (?obj  ?room ?gripper)
        :duration (= ?duration 1)
        :condition  (and  (over all (ball ?obj))
                          (over all (room ?room))
                          (over all (gripper ?gripper))
                          (over all (at-robby ?room))
                          (at start (carry ?obj ?gripper))
                    )
        :effect (and (at end (at ?obj ?room))
                     (at end (free ?gripper))
                     (at start (not (carry ?obj ?gripper)))
                )
    )

)
```

Description des actions hiérarchiques pour le domaine *gripper*

```
(define (domain-helper gripper-strips)
  (:options :abstractOnly :nonConcurentAbstract)
  (:allowed-actions move)

  (:action move-balls
    :parameters (?obj1 ?obj2 ?from ?to)
    :duration (= ?duration 3)
    :conflict-with (free *) (at-robby *) (at ?obj1 *) (at ?obj2 *)
    :precondition (and  (room ?from) (room ?to) (ball ?obj1) (ball ?obj2) (
        at-robby ?from) (at ?obj1 ?from) (at ?obj2 ?from) (free right) (free
        left))
    :effect (and (at-robby ?to) (not (at-robby ?from)) (not (at ?obj1 ?from)
        ) (at ?obj1 ?to) (not (at ?obj2 ?from)) (at ?obj2 ?to) (free left) (
        free right))

    :methods ( :method move-balls-meth1
      :actions (pick1 (pick ?obj1 ?from left))
               (pick2 (pick ?obj2 ?from right))
               (move (move ?from ?to))
               (drop1 (drop ?obj1 ?to left))
               (drop2 (drop ?obj2 ?to right))
      :precondition (!= ?from ?to) (!= ?obj1 ?obj2)
      :causal-links (:init pick1 (ball ?obj1)) (:init pick1 (room ?from)) (:
          init pick1 (gripper left))
                    (:init pick1 (at ?obj1 ?from)) (:init pick1 (at-robby ?
                        from)) (:init pick1 (free left))
                    (:init pick2 (ball ?obj2)) (:init pick2 (room ?from)) (:
                        init pick2 (gripper right))
                    (:init pick2 (at ?obj2 ?from)) (:init pick2 (at-robby ?
                        from)) (:init pick2 (free right))
                    (:init move (room ?from)) (:init move (room ?to)) (:init
                         move (at-robby ?from))
                    (:init drop1 (ball ?obj1)) (:init drop1 (room ?to)) (:
                        init drop1 (gripper left))
                    (:init drop2 (ball ?obj2)) (:init drop2 (room ?to)) (:
                        init drop2 (gripper right))

                    (pick1 drop1 (carry ?obj1 left)) (move drop1 (at-robby ?
                        to))
                    (pick2 drop2 (carry ?obj2 right)) (move drop2 (at-robby
                        ?to))
```

```
29
30                      (pick1 :goal (not (at ?obj1 ?from))) (pick2 :goal (not (
                            at ?obj2 ?from)))
31                      (move :goal (at-robby ?to)) (move :goal (not (at-robby ?
                            from)))
32                      (drop2 :goal (at ?obj2 ?to)) (drop2 :goal (free right))
33      :temporal-links (pick1 move) (pick2 move))
34   )
35 )
```

Description d'un problème du domaine *gripper*

```
(define (problem strips-gripper-x-1)
   (:domain gripper-strips)
   (:objects rooma roomb ball4 ball3 ball2 ball1 left right)
   (:init (room rooma)
          (room roomb)
          (ball ball4)
          (ball ball3)
          (ball ball2)
          (ball ball1)
          (at-robby rooma)
          (free left)
          (free right)
          (at ball4 rooma)
          (at ball3 rooma)
          (at ball2 rooma)
          (at ball1 rooma)
          (gripper left)
          (gripper right))
   (:goal (and (at ball4 roomb)
               (at ball3 roomb)
               (at ball2 roomb)
               (at ball1 roomb))))
```

Description du domaine *blocks*

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; 4 Op-blocks world
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain blocks)
  (:requirements :strips :durative-actions)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x)
               )

  (:durative-action pick-up
            :parameters (?x)
            :duration (= ?duration 1)
            :condition (and (over all (clear ?x))
                            (over all (ontable ?x))
                            (over all (handempty))
                       )
            :effect
            (and  (at end (not (ontable ?x)))
```

```
                    (at end (not (clear ?x)))
                    (at end (not (handempty)))
                    (at end (holding ?x)))
            )

(:durative-action put-down
            :parameters (?x)
            :duration (= ?duration 1)
            :condition (over all (holding ?x))
            :effect
            (and  (at end (not (holding ?x)))
                  (at end (clear ?x))
                  (at end (handempty))
                  (at end (ontable ?x))))

  (:durative-action stack
            :parameters (?x ?y)
            :duration (= ?duration 1)
            :condition (and (over all (holding ?x))
                            (over all (clear ?y))
                       )
            :effect
            (and (at end (not (holding ?x)))
                  (at end (not (clear ?y)))
                  (at end (clear ?x))
                  (at end (handempty))
                  (at end (on ?x ?y))))

  (:durative-action unstack
            :parameters (?x ?y)
            :duration (= ?duration 1)
            :condition (and (over all (on ?x ?y))
                            (over all (clear ?x))
                            (over all (handempty)))
            :effect
            (and  (at end (holding ?x))
                  (at end (clear ?y))
                  (at end (not (clear ?x)))
                  (at end (not (handempty)))
                  (at end (not (on ?x ?y)))))


)
```

Description des actions hiérarchiques pour le domaine *blocks*

```
(define (domain-helper blocks)
  (:options :nonConcurentAbstract)
  (:allowed-actions stack-block-free-pick stack-block-free-unstack clear-
      block-from)
  (:low-priority-predicates handempty)

  /* Stack ?x on ?y if both are free */
  (:action stack-block-free-pick
    :parameters (?x ?y)
    :conflict-with (handempty) (holding *) (on * ?y) (on ?x *) (clear ?y) (
        clear ?x)
    :precondition (and (handempty) (clear ?y) (clear ?x) (ontable ?x))
    :effect (and (on ?x ?y) (handempty) (not (ontable ?x)) (clear ?x))

```

```
13      :methods (:method stack-block-free-1
14                :actions (pick (pick-up ?x))
15                         (put (stack ?x ?y))
16                :duration (= ?duration 2)
17                :precondition (!= ?x ?y)
18                :causal-links
19                  (:init pick (clear ?x)) (:init pick (handempty)) (:init pick
                        (ontable ?x))
20                  (pick put (holding ?x)) (:init put (clear ?y))
21                  (put :goal (clear ?x)) (put :goal (on ?x ?y)) (put :goal (
                        handempty)) (pick :goal (not (ontable ?x)))
22                :temporal-links
23               )
24    )
25
26    /* Stack ?x on ?y if both are free and ?x is on ?z */
27    (:action stack-block-free-unstack
28      :parameters (?x ?y ?z)
29      :conflict-with (handempty) (holding *) (on * ?y) (on ?x *) (clear ?y) (
            clear ?x)
30      :precondition (and (handempty) (clear ?y) (clear ?x) (on ?x ?z))
31      :effect (and (on ?x ?y) (handempty) (clear ?x) (clear ?z) (not (on ?x ?z
            )))
32
33      :methods (:method stack-block-free-1
34                :actions (pick (unstack ?x ?z))
35                         (put (stack ?x ?y))
36                :duration (= ?duration 2)
37                :precondition (!= ?x ?y) (!= ?x ?z) (!= ?z ?y)
38                :causal-links
39                  (:init pick (clear ?x)) (:init pick (handempty)) (:init pick
                        (on ?x ?z))
40                  (pick put (holding ?x)) (:init put (clear ?y))
41                  (pick :goal (clear ?z)) (pick :goal (not (on ?x ?z)))
42                  (put :goal (clear ?x)) (put :goal (on ?x ?y)) (put :goal (
                        handempty))
43                :temporal-links
44               )
45    )
46
47
48    /* Used if y is on top of x. Will put y on the table to clear x */
49    (:action clear-block-from
50      :parameters (?x ?y)
51      :conflict-with (handempty) (holding *) (on ?y *) (on * ?x) (on * ?y) (
            clear ?y)
52      :precondition (and (handempty) (on ?y ?x) (clear ?y))
53      :effect (and (clear ?x) (ontable ?y) (not (on ?y ?x)) (handempty) (clear
            ?y))
54
55      :methods (:method clear-block-from-simple
56                :actions (pick (unstack ?y ?x))
57                         (put (put-down ?y))
58                :duration (= ?duration 2)
59                :precondition (!= ?x ?y)
60                :causal-links
61                  (:init pick (on ?y ?x)) (:init pick (clear ?y)) (:init pick
                        (handempty))
62                  (pick put (holding ?y))
```

```
                (pick :goal (clear ?x)) (put :goal (ontable ?y))
                (pick :goal (not (on ?y ?x))) (put :goal (handempty)) (put :
                    goal (clear ?y))
              :temporal-links
            )
  )
)
```

Description d'un problème du domaine *blocks*

```
(define (problem BLOCKS-4-0)
(:domain BLOCKS)
(:objects D B A C )
(:INIT (CLEAR C) (CLEAR A) (CLEAR B) (CLEAR D)
 (ONTABLE C) (ONTABLE A)
 (ONTABLE B) (ONTABLE D) (HANDEMPTY))
(:goal (AND (ON D C) (ON C B) (ON B A)))
)
```

Description du domaine *satellite*

```
(define (domain satellite)
  (:requirements :strips :typing :durative-actions)
  (:types satellite direction instrument mode)
  (:predicates
    (on_board ?i - instrument ?s - satellite)
    (supports ?i - instrument ?m - mode)
    (pointing ?s - satellite ?d - direction)
    (power_avail ?s - satellite)
    (power_on ?i - instrument)
    (calibrated ?i - instrument)
    (have_image ?d - direction ?m - mode)
    (calibration_target ?i - instrument ?d - direction))

  (:durative-action turn_to
   :parameters (?s - satellite
                ?d_new - direction ?d_prev - direction)
   :duration (= ?duration 1)
   :condition (and (over all (pointing ?s ?d_prev)))
   :effect (and  (at end (pointing ?s ?d_new))
                 (at end (not (pointing ?s ?d_prev)))
           )
  )

  (:durative-action switch_on
   :parameters (?i - instrument ?s - satellite)
   :duration (= ?duration 1)
   :condition (and (over all (on_board ?i ?s))
                   (over all (power_avail ?s))
              )
   :effect (and (at end (power_on ?i))
                (at end (not (calibrated ?i)))
                (at end (not (power_avail ?s)))
           )
  )

  (:durative-action switch_off
   :parameters (?i - instrument ?s - satellite)
   :duration (= ?duration 1)
```

```
    :condition (and (over all (on_board ?i ?s))
                    (over all (power_on ?i))
               )
    :effect (and (at end (not (power_on ?i)))
                 (at end (power_avail ?s))
            )
  )

  (:durative-action calibrate
   :parameters (?s - satellite ?i - instrument ?d - direction)
   :duration (= ?duration 1)
   :condition (and (over all (on_board ?i ?s))
                   (over all (calibration_target ?i ?d))
                   (over all (pointing ?s ?d))
                   (over all (power_on ?i))
               )
   :effect (at end (calibrated ?i))
  )

  (:durative-action take_image
   :parameters (?s - satellite ?d - direction
                ?i - instrument ?m - mode)
   :duration (= ?duration 1)
   :condition (and (over all (calibrated ?i))
                   (over all (on_board ?i ?s))
                   (over all (supports ?i ?m))
                   (over all (power_on ?i))
                   (over all (pointing ?s ?d))
                   (over all (power_on ?i))
               )
   :effect (at end (have_image ?d ?m))
  )
)
```

Description des actions hiérarchiques pour le domaine *satellite*

```
(define (domain-helper satellite     )
  (:options :abstractOnly)
  (:allowed-actions switch_off)

  (:action move
    :parameters (?s - satellite ?d_new - direction)
    :duration (= ?duration 1)
    :conflict-with (pointing ?s *)
    :precondition (and )
    :effect (and (pointing ?s ?d_new))

    :methods ( :method move-1
               :parameters (?d_prev - direction)
               :actions (turn (turn_to ?s ?d_new ?d_prev))
               :precondition (!= ?d_new ?d_prev) (pointing ?s ?d_prev)
               :causal-links
                     (:init turn (pointing ?s ?d_prev)) (turn :goal (
                         pointing ?s ?d_new))
               :temporal-links
             )
  )

  (:action activate
    :parameters (?i - instrument ?s - satellite ?d_cal - direction)
```

```
24      :duration (= ?duration 2)
25      :conflict-with (power_avail ?s) (calibrated ?i) (pointing ?s *)
26      :precondition  (and (on_board ?i ?s)
27                          (power_avail ?s)
28                                  (calibration_target ?i ?d_cal))
29      :effect (and
30                  (power_on ?i)
31                  (not (power_avail ?s))
32                  (calibrated ?i))
33      :side-effect (and (pointing ?s ?d_cal))
34
35      :methods ( :method activate-1
36                 :actions (turn (move ?s ?d_cal))
37                          (switch (switch_on ?i ?s))
38                          (calib (calibrate ?s ?i ?d_cal))
39                 :precondition
40                 :causal-links
41
42
43                      (:init switch (on_board ?i ?s)) (:init switch (
                            power_avail ?s))
44
45                      (:init calib (on_board ?i ?s)) (:init calib (
                            calibration_target ?i ?d_cal))
46                      (turn calib (pointing ?s ?d_cal)) (switch calib (
                            power_on ?i))
47
48                      (turn :goal (pointing ?s ?d_cal))
49                      (switch :goal (power_on ?i)) (switch :goal (not (
                            power_avail ?s)))
50                      (calib :goal (calibrated ?i))
51                 :temporal-links
52               )
53  )
54
55  (:action turn-take-picture
56    :parameters (?s - satellite ?d_new - direction ?i - instrument ?m - mode
          )
57    :duration (= ?duration 2)
58    :conflict-with (pointing ?s *)
59    :precondition (and (calibrated ?i)
60                       (on_board ?i ?s)
61                       (supports ?i ?m)
62                       (power_on ?i))
63    :effect (and (have_image ?d_new ?m) )
64    :side-effect (and (pointing ?s ?d_new) )
65
66      :methods ( :method turn-take-picture-1
67                 :actions (turn (move ?s ?d_new))
68                          (take (take_image ?s ?d_new ?i ?m))
69                 :precondition
70                 :causal-links
71                          (:init take (calibrated ?i))
72                          (:init take (on_board ?i ?s))
73                          (:init take (supports ?i ?m))
74                          (:init take (power_on ?i))
75                          (:init take (power_on ?i))
76                          (turn take (pointing ?s ?d_new))
77
```

```
                        (turn :goal (pointing ?s ?d_new))
                        (take :goal (have_image ?d_new ?m))
                  :temporal-links
                )
  )
)
```

Description d'un problème du domaine *satellite*

```
(define (problem strips-sat-x-1)
(:domain satellite)
(:objects
        satellite0 - satellite
        instrument0 - instrument
        image1 - mode
        spectrograph2 - mode
        thermograph0 - mode
        Star0 - direction
        GroundStation1 - direction
        GroundStation2 - direction
        Phenomenon3 - direction
        Phenomenon4 - direction
        Star5 - direction
        Phenomenon6 - direction
)
(:init
        (supports instrument0 thermograph0)
        (calibration_target instrument0 GroundStation2)
        (on_board instrument0 satellite0)
        (power_avail satellite0)
        (pointing satellite0 Phenomenon6)
)
(:goal (and
        (have_image Phenomenon4 thermograph0)
        (have_image Star5 thermograph0)
        (have_image Phenomenon6 thermograph0)
))

)
```

Description du domaine *logistics*

```
;; logistics domain
;;

(define (domain logistics-modified)
  (:requirements :strips :durative-actions)
  (:predicates  (package ?obj)
                (truck ?truck)
                (airplane ?airplane)
                (airport ?airport)
                (location ?loc)
                (in-city ?obj ?city)
                (city ?city)
                (at ?obj ?loc)
                (in ?obj ?v)
                (free ?v))
```

```
(:durative-action load-truck
  :parameters
   (?obj
    ?truck
    ?loc)
       :duration (= ?duration 1)
  :condition
   (and (over all (package ?obj))
        (over all (truck ?truck))
        (over all (location ?loc))
        (over all (at ?truck ?loc))
        (over all (at ?obj ?loc))
        (over all (free ?truck)))
  :effect
   (and (at end (not (at ?obj ?loc)))
        (at end (in ?obj ?truck))
        (at end (not (free ?truck))) ))

(:durative-action load-airplane
  :parameters
   (?obj
    ?airplane
    ?loc)
       :duration (= ?duration 1)
  :condition
   (and (over all (package ?obj))
        (over all (airplane ?airplane))
        (over all (location ?loc))
        (over all (at ?obj ?loc))
        (over all (at ?airplane ?loc))
        (over all (free ?airplane)))
  :effect
   (and (at end (not (at ?obj ?loc)))
        (at end (in ?obj ?airplane))
        (at end (not (free ?airplane))) ))

(:durative-action unload-truck
  :parameters
   (?obj
    ?truck
    ?loc)
       :duration (= ?duration 1)
  :condition
   (and (over all (package ?obj))
        (over all (truck ?truck))
        (over all (location ?loc))
        (over all (at ?truck ?loc))
        (over all (in ?obj ?truck)))
  :effect
   (and (at end (not (in ?obj ?truck)))
        (at end (at ?obj ?loc))
        (at end (free ?truck)) ))

(:durative-action unload-airplane
  :parameters
   (?obj
    ?airplane
    ?loc)
       :duration (= ?duration 1)
```

```
  :condition
   (and (over all (package ?obj))
        (over all (airplane ?airplane))
        (over all (location ?loc))
        (over all (in ?obj ?airplane))
        (over all (at ?airplane ?loc)))
  :effect
   (and (at end (not (in ?obj ?airplane)))
        (at end (at ?obj ?loc))
        (at end (free ?airplane))))

(:durative-action drive-truck
  :parameters
   (?truck
    ?loc-from
    ?loc-to
    ?city)
       :duration (= ?duration 1)
  :condition
   (and (over all (truck ?truck))
        (over all (location ?loc-from))
        (over all (location ?loc-to))
        (over all (city ?city))
        (over all (at ?truck ?loc-from))
        (over all (in-city ?loc-from ?city))
        (over all (in-city ?loc-to ?city)))
  :effect
   (and (at end (not (at ?truck ?loc-from)))
        (at end (at ?truck ?loc-to))))

(:durative-action fly-airplane
  :parameters
   (?airplane
    ?loc-from
    ?loc-to)
  :duration (= ?duration 1)
  :condition
   (and (over all (airplane ?airplane))
        (over all (airport ?loc-from))
        (over all (airport ?loc-to))
        (over all (at ?airplane ?loc-from)))
  :effect
   (and (at end (not (at ?airplane ?loc-from)))
        (at end (at ?airplane ?loc-to))))
)
```

Description des actions hiérarchiques pour le domaine *logistics*

```
(define (domain-helper logistics-modified)
  (:options :abstractOnly)
  (:allowed-actions drive-truck fly-airplane)

  (:action same-city-delivery
    :parameters (?obj ?truck ?from ?to ?city)
    :conflict-with (at ?obj *) (at ?truck *) (free ?truck)
    :precondition (and  (package ?obj) (truck ?truck) (location ?from) (
        location ?to) (city ?city)
                        (at ?truck ?from) (at ?obj ?from) (in-city ?from ?
                            city) (in-city ?to ?city) (free ?truck) )
```

```
10      :effect (and (at ?obj ?to) (not (at ?obj ?from)) (free ?truck) (at ?
            truck ?to) (not (at ?truck ?from)))
11      :side-effect (and )
12
13      :methods ( :method same-city-delivery-different-loc
14                 :actions (load (load-truck ?obj ?truck ?from))
15                          (move (drive-truck ?truck ?from ?to ?city))
16                          (unload (unload-truck ?obj ?truck ?to))
17                 :duration (= ?duration 3)
18                 :precondition (!= ?from ?to)
19                 :causal-links (:init load (package ?obj)) (:init load (truck
                       ?truck)) (:init load (location ?from))
20                               (:init load (at ?truck ?from)) (:init load (at
                                   ?obj ?from)) (:init load (free ?truck))
21
22                               (:init move (truck ?truck)) (:init move (
                                   location ?from)) (:init move (location ?to))
                                    (:init move (city ?city))
23                               (:init move (at ?truck ?from)) (:init move (in-
                                   city ?from ?city)) (:init move (in-city ?to
                                   ?city))
24
25                               (:init unload (package ?obj)) (:init unload (
                                   truck ?truck)) (:init unload (location ?to))
26                               (move unload (at ?truck ?to)) (load unload (in
                                   ?obj ?truck))
27
28                               (unload :goal (at ?obj ?to)) (load :goal (not (
                                   at ?obj ?from))) (unload :goal (free ?truck)
                                   )
29                               (move :goal (at ?truck ?to)) (move :goal (not (
                                   at ?truck ?from)))
30
31                     :temporal-links (load move)
32              )
33  )
34
35  (:action airport-delivery
36      :parameters (?obj ?airplane ?from ?to)
37      :duration (= ?duration 3)
38      :conflict-with (at ?obj *) (at ?airplane *) (free ?airplane)
39      :precondition (and  (package ?obj) (airplane ?airplane) (location ?
            from) (location ?to)  (airport ?from) (airport ?to)
40                          (at ?airplane ?from) (at ?obj ?from) (free ?
                                airplane))
41      :effect (and (at ?obj ?to) (not (at ?obj ?from)) (free ?airplane) (not
             (at ?airplane ?from)) (at ?airplane ?to))
42      :side-effect (and )
43
44      :methods ( :method airport-delivery-meth1
45                 :actions (load (load-airplane ?obj ?airplane ?from)) (
                       move (fly-airplane ?airplane ?from ?to)) (unload (
                       unload-airplane ?obj ?airplane ?to))
46                  :precondition (!= ?from ?to)
47                 :causal-links (:init load (package ?obj)) (:init load (
                       airplane ?airplane)) (:init load (location ?from))
48                        (:init load (at ?airplane ?from)) (:init load
                              (at ?obj ?from)) (:init load (free ?
                              airplane))
```

```
49
50                                  (:init move (airplane ?airplane)) (:init move
                                        (airport ?from)) (:init move (airport ?to))
51                                  (:init move (at ?airplane ?from))
52
53                                  (:init unload (package ?obj)) (:init unload (
                                        airplane ?airplane)) (:init unload (
                                        location ?to))
54                                  (move unload (at ?airplane ?to)) (load unload
                                        (in ?obj ?airplane))
55
56                                  (load :goal (not (at ?obj ?from))) (unload :
                                        goal (at ?obj ?to)) (unload :goal (free ?
                                        airplane))
57                                  (move :goal (not (at ?airplane ?from))) (move
                                        :goal (at ?airplane ?to))
58
59                      :temporal-links (load move))
60    )
61 )
```

Description d'un problème du domaine *logistics*

```
(define (problem logistics-modified-prb001)
        (:domain logistics-modified)
        (:objects
                city1 apt1 loc1
5               city2 apt2 loc2
                city3 apt3 loc3
                apn1
                truck1 truck2 truck3
                p-apt1-loc2 p-apt3-loc2 p-apt1-loc1
10              p-loc3-loc1 p-apt2-apt1
        )

        (:init
                (package p-apt1-loc2) (package p-apt3-loc2)
15              (package p-apt1-loc1)   (package p-loc3-loc1)
                (package p-apt2-apt1)
                (truck truck1) (truck truck2) (truck truck3)
                (airplane apn1)
                (free truck1) (free truck2) (free truck3)
20              (free apn1)
                (city city1) (airport apt1) (location apt1)
                (in-city apt1 city1) (location loc1)
                (in-city loc1 city1)
                (city city2) (airport apt2) (location apt2)
25              (in-city apt2 city2) (location loc2)
                (in-city loc2 city2)
                (city city3) (airport apt3) (location apt3)
                (in-city apt3 city3) (location loc3)
                (in-city loc3 city3)
30
                (at apn1 apt1)
                (at truck1 loc1) (at truck2 loc2)
                (at truck3 loc3)
                (at p-apt1-loc2 apt1) (at p-apt3-loc2 apt3)
35              (at p-apt1-loc1 apt1) (at p-loc3-loc1 loc3)
                (at p-apt2-apt1 apt2)
        )
```

```
        (:goal (and
                (at p-apt1-loc2 loc2)
                (at p-apt3-loc2 loc2)
                (at p-apt1-loc1 loc1)
                (at p-loc3-loc1 loc1)
                (at p-apt2-apt1 apt1)
        ))

)
```

Description du domaine *survivors*

```
(define (domain survivor)
  (:requirements :strips :typing :durative-actions :equality)
  (:types robot survivor loc zone team)

  (:predicates
    (explored ?z - loc)
    (at-r ?r - robot ?z - loc)
    (at-s ?s - survivor ?z - loc)
    (stabilized ?s - survivor)
    (hospitalized ?s - survivor)
    (belong ?l - loc ?z - zone)
    (adjacent ?l1 ?l2 - loc)
    (part-of ?r - robot ?t - team)
    (different ?r1 ?r2 - robot)
    (at-team ?t - team ?z - zone)
    (hospital ?l - loc)
  )

  (:functions
    (distance ?from ?to - loc)
    (distance-zone ?from ?to - zone)
  )

  (:durative-action move
    :parameters (?r - robot ?from ?to - loc)
    :duration (= ?duration (distance ?from ?to))
    :condition (and (at start (at-r ?r ?from))
                    (over all (not (= ?from ?to))) )
    :effect (and (at end (at-r ?r ?to))
                 (at start (not (at-r ?r ?from))) ))

  (:durative-action explore
    :parameters (?r - robot ?l - loc)
    :duration (= ?duration 1)
    :condition (and (over all (at-r ?r ?l)) )
    :effect (and (at end (explored ?l)) ))

  (:durative-action move-team
    :parameters (?team - team ?from ?to - zone)
    :duration (= ?duration (distance-zone ?from ?to))
    :condition (and (at start (at-team ?team ?from))
                    (over all (not (= ?from ?to))) )
    :effect (and (at start (not (at-team ?team ?from)))
                 (at end (at-team ?team ?to)) )
  )
```

```
  (:durative-action stabilize
    :parameters (?r - robot ?s - survivor ?l - loc)
    :duration (= ?duration 1)
    :condition (and (over all (at-r ?r ?l))
                    (over all (at-s ?s ?l)) )
    :effect (and (at end (stabilized ?s)) )
  )

  (:durative-action hospitalize
    :parameters (?s - survivor ?l - loc)
    :duration (= ?duration 1)
    :condition (and (over all (stabilized ?s))
                    (over all (hospital ?l))
                    (over all (at-s ?s ?l)) )
    :effect (and (at end (hospitalized ?s)) )
  )

  (:durative-action move-survivor
    :parameters (?r1 ?r2 - robot ?s - survivor ?from ?to - loc)
    :duration (= ?duration (distance ?from ?to))
    :condition (and (over all (stabilized ?s))
                    (at start (at-r ?r1 ?from))
                    (at start (at-r ?r2 ?from))
                    (at start (at-s ?s ?from))
                    (over all (different ?r1 ?r2))
                    (over all (not (= ?from ?to))))
    :effect (and (at end (at-r ?r1 ?to))
                 (at end (at-r ?r2 ?to))
                 (at end (at-s ?s ?to))
                 (at start (not (at-r ?r1 ?from)))
                 (at start (not (at-r ?r2 ?from)))
                 (at start (not (at-s ?s ?from))) )
  )
)
```

Description des actions hiérarchiques pour un problème du domaine *survivors*. A cause de contraintes inhérentes à PDDL, certaines actions ont été partiellement instanciées dès la description. Une seule action de chaque type est présentée ici.

```
(define (domain-helper survivor)
        (:options :erasePlansWhenAbstractMet)
        (:allowed-actions move move-team treat-survivor-team1-l3_1 treat-
            survivor-team2-l3_1 treat-survivor-team1-l0_2 treat-survivor-
            team2-l0_2 explore-z0_0-team1 explore-z0_0-team2 explore-z0_1-
            team1 explore-z0_1-team2 explore-z1_0-team1 explore-z1_0-team2
            explore-z1_1-team1 explore-z1_1-team2 )

        (:low-priority-predicates at-r)

  (:action treat-survivor-team1-l3_1

    :parameters(?s - survivor ?l - loc ?z - zone)
    :conflict-with (at-team team1 *) (at-r r1_1 *) (move r1_1 * *) (at-r
        r1_2 *) (move r1_2 * *) (at-s ?s *)
    :precondition (and (at-team team1 ?z) (belong ?l ?z) (at-s ?s ?l))
    :effect (and (hospitalized ?s))
    :side-effect (and (at-team team1 z1_0))
    :methods (
      :method hospitalize-to-l3_1
```

```
16          :actions
17            (stab (stabilize r1_1 ?s ?l))
18            (m (move-to-hospital-team1 ?s ?l l3_1 ?z z1_0))
19            (hos (hospitalize ?s l3_1))
20          :duration (= ?duration  1)
21          :precondition
22          :causal-links
23            (:init stab (at-s ?s ?l))
24            (:init m (at-s ?s ?l))
25            (:init m (at-team team1 ?z))
26            (stab m (stabilized ?s))
27            (stab hos (stabilized ?s))
28            (m hos (at-s ?s l3_1))
29            (hos :goal (hospitalized ?s))
30            (m :goal (at-team team1 z1_0))
31          :temporal-links
32      )
33    )
34
35    (...)
36
37    (:action move-to-hospital-team1
38            :parameters(?s - survivor ?from ?to - loc ?z1 ?z2 - zone)
39            :conflict-with (at-team team1 *) (at-s ?s *) (at-r r1_1 *) (at-r
                r1_2 *)
40            :precondition (and (hospital ?to) (at-s ?s ?from) (stabilized ?s)
                (at-team team1 ?z1) (belong ?from ?z1) (belong ?to ?z2))
41            :effect (and (at-s ?s ?to))
42            :side-effect (and (at-team team1 ?z2))
43            :methods (
44                    :method move-to-hospital-team1-True
45                    :actions
46                            (m (move-survivor r1_1 r1_2 ?s ?from ?to))
47                    :duration (= ?duration  (distance ?from ?to))
48                    :precondition (!= ?from ?to) (at-r r1_1 ?from) (at-r r1_2
                        ?from)(= ?z1 ?z2)
49                    :causal-links
50                            (:init m (at-r r1_1 ?from))
51                            (:init m (at-r r1_2 ?from))
52                            (:init m (stabilized ?s))
53                            (:init m (at-s ?s ?from))
54                            (m :goal (at-s ?s ?to))
55                            (:init :goal (at-team team1 ?z1))
56                    :temporal-links
57
58                    :method move-to-hospital-team1-False
59                    :actions
60                            (m (move-survivor r1_1 r1_2 ?s ?from ?to))
61                            (m-team (move-team team1 ?z1 ?z2))
62                    :duration (= ?duration  (distance ?from ?to))
63                    :precondition (!= ?from ?to) (at-r r1_1 ?from) (at-r r1_2
                        ?from)(!= ?z1 ?z2)
64                    :causal-links
65                            (:init m (at-r r1_1 ?from))
66                            (:init m (at-r r1_2 ?from))
67                            (:init m (stabilized ?s))
68                            (:init m (at-s ?s ?from))
69                            (m :goal (at-s ?s ?to))
70                            (:init m-team (at-team team1 ?z1))
```

```
                        (m-team :goal (at-team team1 ?z2))
                :temporal-links

        )
    )

    (...)

    (:action explore-z0_0-team1
        :parameters ()
        :conflict-with (at-team team1 *) (at-r r1_1 *) (at-r r1_2 *)
        :precondition (and (at-team team1 z0_0))
        :effect (and (explored l0_0) (explored l0_1) (explored l1_0) (
            explored l1_1) )
        :side-effect (and (at-team team1 z0_0))
        :methods (
            :method explore-zone-1
            :actions
                (move-0-1 (move r1_1 l0_1 l0_0))
                (explore-0-1 (explore r1_1 l0_1))
                (explore-1-1 (explore r1_1 l0_0))
                (move-0-2 (move r1_2 l1_0 l1_1))
                (explore-0-2 (explore r1_2 l1_0))
                (explore-1-2 (explore r1_2 l1_1))
            :duration (= ?duration  3)
            :precondition
            :causal-links
                (:init :goal (at-team team1 z0_0))
                (explore-0-1 :goal (explored l0_1))
                (explore-1-1 :goal (explored l0_0))
                (move-0-1 explore-1-1 (at-r r1_1 l0_0))
                (explore-0-2 :goal (explored l1_0))
                (explore-1-2 :goal (explored l1_1))
                (move-0-2 explore-1-2 (at-r r1_2 l1_1))
            :temporal-links
                (explore-0-1 move-0-1)
                (explore-0-2 move-0-2)
        )
    )

    (...)
)
```

Description d'un problème du domaine *survivors*. Certains fluent répétitifs ont été omis (en particulier les distances entre cellules).

```
(define (problem survivor-p001)
        (:domain survivor)

        (:objects
                r1_1 r1_2 r2_1 r2_2 - robot
                s1 s2 - survivor
                team1 team2 - team
                l0_0 l0_1 l0_2 l0_3 l1_0 l1_1 l1_2 l1_3 l2_0 l2_1 l2_2 l2_3
                    l3_0 l3_1 l3_2 l3_3 - loc
                z0_0 z0_1 z1_0 z1_1 - zone
        )

        (:init
```

```
13                (belong l0_0 z0_0)
14                (belong l0_1 z0_0)
15                (belong l1_0 z0_0)
16                (belong l1_1 z0_0)
17                (belong l0_2 z0_1)
18                (belong l0_3 z0_1)
19                (belong l1_2 z0_1)
20                (belong l1_3 z0_1)
21                (belong l2_0 z1_0)
22                (belong l2_1 z1_0)
23                (belong l3_0 z1_0)
24                (belong l3_1 z1_0)
25                (belong l2_2 z1_1)
26                (belong l2_3 z1_1)
27                (belong l3_2 z1_1)
28                (belong l3_3 z1_1)
29
30                (different r1_1 r1_2)
31                (different r1_1 r2_1)
32                (different r1_1 r2_2)
33                (different r1_2 r2_1)
34                (different r1_2 r2_2)
35                (different r2_1 r2_2)
36
37                (part-of r1_1 team1)
38                (part-of r1_2 team1)
39                (part-of r2_1 team2)
40                (part-of r2_2 team2)
41
42                (at-r r1_1 l0_0) (at-r r1_2 l0_0) (at-r r2_1 l0_0) (at-r
                      r2_2 l0_0)
43
44                (at-team team1 z0_0) (at-team team2 z0_0)
45
46                (at-s s1 l0_3) (at-s s2 l3_2)
47
48                (hospital l3_1) (hospital l0_2)
49
50                (= (distance l0_0 l0_0) 0.0)
51                (= (distance l0_0 l0_1) 1.0)
52                (= (distance l0_0 l0_2) 2.0)
53                (= (distance l0_0 l0_3) 3.0)
54                (= (distance l0_0 l1_0) 1.0)
55                (= (distance l0_0 l1_1) 2.0)
56                (= (distance l0_0 l1_2) 3.0)
57                (= (distance l0_0 l1_3) 4.0)
58                (= (distance l0_0 l2_0) 2.0)
59                (= (distance l0_0 l2_1) 3.0)
60                (= (distance l0_0 l2_2) 4.0)
61                (= (distance l0_0 l2_3) 5.0)
62                (= (distance l0_0 l3_0) 3.0)
63                (= (distance l0_0 l3_1) 4.0)
64                (= (distance l0_0 l3_2) 5.0)
65                (= (distance l0_0 l3_3) 6.0)
66                (= (distance l0_1 l0_0) 1.0)
67                (...)
68                (= (distance l3_3 l3_2) 1.0)
69                (= (distance l3_3 l3_3) 0.0)
70                (= (distance-zone z0_0 z0_0) 0.0)
```

```
71                  (= (distance-zone z0_0 z0_1) 1.0)
72                  (= (distance-zone z0_0 z1_0) 1.0)
73                  (= (distance-zone z0_0 z1_1) 1.83)
74                  (= (distance-zone z0_1 z0_0) 1.0)
75                  (= (distance-zone z0_1 z0_1) 0.0)
76                  (= (distance-zone z0_1 z1_0) 1.83)
77                  (= (distance-zone z0_1 z1_1) 1.0)
78                  (= (distance-zone z1_0 z0_0) 1.0)
79                  (= (distance-zone z1_0 z0_1) 1.83)
80                  (= (distance-zone z1_0 z1_0) 0.0)
81                  (= (distance-zone z1_0 z1_1) 1.0)
82                  (= (distance-zone z1_1 z0_0) 1.83)
83                  (= (distance-zone z1_1 z0_1) 1.0)
84                  (= (distance-zone z1_1 z1_0) 1.0)
85                  (= (distance-zone z1_1 z1_1) 0.0)
86          )
87
88          (:goal (and
89                  (explored l0_0)
90                  (explored l0_1)
91                  (explored l0_2)
92                  (explored l0_3)
93                  (explored l1_0)
94                  (explored l1_1)
95                  (explored l1_2)
96                  (explored l1_3)
97                  (explored l2_0)
98                  (explored l2_1)
99                  (explored l2_2)
100                 (explored l2_3)
101                 (explored l3_0)
102                 (explored l3_1)
103                 (explored l3_2)
104                 (explored l3_3)
105
106                 (hospitalized s1) (hospitalized s2)
107         ))
108
109 )
```

C

Description en PDDL de l'exemple du chapitre 4

Description du domaine

```
(define (domain action-ex-pddl2)
  (:requirements :strips :typing :durative-actions)
  (:types robot - object
          aav agv - robot
          loc - object)

  (:predicates
    (explored ?z - loc)
    (at ?r - robot ?z - loc)
    (adjacent ?z1 ?z2 - loc)
    (aav-allowed ?z - loc)
    (agv-allowed ?z - loc)
  )

  (:functions
    (distance-aav ?from ?to - loc)
    (distance-agv ?from ?to - loc)
  )

  (:durative-action move-aav
    :parameters (?r - aav ?from ?to - loc)
    :duration (= ?duration (distance-aav ?from ?to))
    :condition (and (at start (at ?r ?from))
                    (over all (aav-allowed ?from))
                    (over all (aav-allowed ?to))
                    (over all (adjacent ?from ?to)))
    :effect (and (at end (at ?r ?to))
                 (at start (not (at ?r ?from))) )
  )

  (:durative-action move-agv
    :parameters (?r - agv ?from ?to - loc)
    :duration (= ?duration (distance-agv ?from ?to))
    :condition (and (at start (at ?r ?from))
                    (over all (agv-allowed ?from))
                    (over all (agv-allowed ?to))
                    (over all (adjacent ?from ?to)))
    :effect (and (at end (at ?r ?to))
                 (at start (not (at ?r ?from))) )
  )
```

```
  (:durative-action explore
    :parameters (?r - robot ?l - loc)
    :duration (= ?duration 1)
    :condition (and (over all (at ?r ?l)) )
    :effect (and (at end (explored ?l)) )
  )
)
```

Description du problème

```
(define (problem no-motion-ex)
  (:domain action-ex-pddl2)

  (:objects
    aav1 - aav
    cell1 cell3 cell4 cell5 - loc
  )

  (:init
    (adjacent cell1 cell3) (adjacent cell3 cell1)
    (adjacent cell1 cell4) (adjacent cell4 cell1)
    (adjacent cell3 cell4) (adjacent cell4 cell3)
    (adjacent cell1 cell5) (adjacent cell5 cell1)
    (adjacent cell3 cell5) (adjacent cell5 cell3)
    (adjacent cell4 cell5) (adjacent cell5 cell4)

    (= (distance-aav cell1 cell3) 2)
    (= (distance-aav cell3 cell1) 2)
    (= (distance-aav cell1 cell4) 3)
    (= (distance-aav cell4 cell1) 3)
    (= (distance-aav cell1 cell5) 4)
    (= (distance-aav cell5 cell1) 4)
    (= (distance-aav cell3 cell4) 1)
    (= (distance-aav cell4 cell3) 1)
    (= (distance-aav cell3 cell5) 2)
    (= (distance-aav cell5 cell3) 2)
    (= (distance-aav cell4 cell5) 1)
    (= (distance-aav cell5 cell4) 1)

    (aav-allowed cell1)
    (aav-allowed cell3)
    (aav-allowed cell4)
    (aav-allowed cell5)

    (at aav1 cell1)
  )

  (:goal (and
    (explored cell3)
    (explored cell4)
    (explored cell5)
  ))
)
```

D

Description en PDDL de l'exemple du chapitre 5

Description du domaine

```
(define (domain ex-repair)
  (:requirements :strips :typing :durative-actions
                 :equality :agents-def)
  (:types robot - object
          aav agv - robot
          loc - object)

  (:predicates
    (explored ?z - loc)
    (at ?r - robot ?z - loc)
    (aav-allowed ?z - loc)
    (agv-allowed ?z - loc)
    (in-com ?r1 ?r2 - robot ?l - loc)
  )

  (:functions
    (distance-aav ?from ?to - loc)
    (distance-agv ?from ?to - loc)
  )

  (:durative-action move-aav
    :parameters (?r - aav ?from ?to - loc)
    :agent (?r)
    :duration (= ?duration (distance-aav ?from ?to))
    :condition (and (at start (at ?r ?from))
                    (over all (aav-allowed ?from))
                    (over all (aav-allowed ?to))
                    (over all (not (= ?from ?to))) )
    :effect (and (at end (at ?r ?to))
                 (at start (not (at ?r ?from))) )
  )

  (:durative-action move-agv
    :parameters (?r - agv ?from ?to - loc)
    :agent (?r)
    :duration (= ?duration (distance-agv ?from ?to))
    :condition (and (at start (at ?r ?from))
                    (over all (agv-allowed ?from))
                    (over all (agv-allowed ?to))
                    (over all (not (= ?from ?to))))
    :effect (and (at end (at ?r ?to))
```

```
                (at start (not (at ?r ?from))) )
  )

  (:durative-action explore
    :parameters (?r - robot ?l - loc)
    :agent (?r)
    :duration (= ?duration 1)
    :condition (and (over all (at ?r ?l)) )
    :effect (and (at end (explored ?l)) )
  )

(:durative-action com-passive
  :parameters (?r1 ?r2 - robot ?l - loc)
  :agent (?r1)
  :duration (= ?duration 1)
  :condition (and (over all (not (= ?r1 ?r2)))
                  (over all (at ?r1 ?l)))
  :effect (and (at start (in-com ?r1 ?r2 ?l))
               (at end (not (in-com ?r1 ?r2 ?l))))
)

(:durative-action com-active
  :parameters (?r1 ?r2 - robot ?l - loc)
  :agent (?r1)
  :duration (= ?duration 0.5)
  :condition (and (over all (not (= ?r1 ?r2)))
                  (over all (in-com ?r1 ?r2 ?l))
                  (over all (in-com ?r2 ?r1 ?l)))
  :effect ()
)

)
```

Description des actions hiérarhichiques

```
(define (domain-helper ex-repair)
    (:options :abstractOnly :erasePlansWhenAbstractMet)
    (:allowed-actions move-aav move-agv com-passive)
    (:low-priority-predicates )


(:action patrol_aav_1
  :parameters (?r - aav)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell11))
  :effect (and (explored cell11) (explored cell21))
  :side-effect (and (at ?r cell21))
  :methods(

      :method patrol_aav_1_m
      :actions (exp0 (explore ?r cell11))
        (move (move-aav ?r cell11 cell21))
        (exp1 (explore ?r cell21))

      :precondition
      :causal-links (:init exp0 (at ?r cell11))
        (:init move (at ?r cell11))
```

```
        (move exp1 (at ?r cell21))
        (move :goal (at ?r cell21))
        (exp0 :goal (explored cell11))
        (exp1 :goal (explored cell21))
      :temporal-links (exp0 move)

  )
)


(:action patrol_aav_2
  :parameters (?r - aav)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell21))
  :effect (and (explored cell21) (explored cell11))
  :side-effect (and (at ?r cell11))
  :methods(

      :method patrol_aav_2_m
      :actions (exp0 (explore ?r cell21))
        (move (move-aav ?r cell21 cell11))
        (exp1 (explore ?r cell11))

      :precondition
      :causal-links (:init exp0 (at ?r cell21))
        (:init move (at ?r cell21))
        (move exp1 (at ?r cell11))
        (move :goal (at ?r cell11))
        (exp0 :goal (explored cell21))
        (exp1 :goal (explored cell11))
      :temporal-links (exp0 move)

  )
)


(:action patrol_aav_3
  :parameters (?r - aav)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell31))
  :effect (and (explored cell31) (explored cell41))
  :side-effect (and (at ?r cell41))
  :methods(

      :method patrol_aav_3_m
      :actions (exp0 (explore ?r cell31))
        (move (move-aav ?r cell31 cell41))
        (exp1 (explore ?r cell41))

      :precondition
      :causal-links (:init exp0 (at ?r cell31))
        (:init move (at ?r cell31))
        (move exp1 (at ?r cell41))
        (move :goal (at ?r cell41))
        (exp0 :goal (explored cell31))
```

```
        (exp1 :goal (explored cell41))
      :temporal-links (exp0 move)

  )
)

(:action patrol_aav_4
  :parameters (?r - aav)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell41))
  :effect (and (explored cell41) (explored cell31))
  :side-effect (and (at ?r cell31))
  :methods(

      :method patrol_aav_4_m
      :actions (exp0 (explore ?r cell41))
        (move (move-aav ?r cell41 cell31))
        (exp1 (explore ?r cell31))

      :precondition
      :causal-links (:init exp0 (at ?r cell41))
        (:init move (at ?r cell41))
        (move exp1 (at ?r cell31))
        (move :goal (at ?r cell31))
        (exp0 :goal (explored cell41))
        (exp1 :goal (explored cell31))
      :temporal-links (exp0 move)

  )
)


(:action patrol_agv_1
  :parameters (?r - agv)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell13))
  :effect (and (explored cell13) (explored cell23))
  :side-effect (and (at ?r cell23))
  :methods(

      :method patrol_agv_1_m
      :actions (exp0 (explore ?r cell13))
        (move (move-agv ?r cell13 cell23))
        (exp1 (explore ?r cell23))

      :precondition
      :causal-links (:init exp0 (at ?r cell13))
        (:init move (at ?r cell13))
        (move exp1 (at ?r cell23))
        (move :goal (at ?r cell23))
        (exp0 :goal (explored cell13))
        (exp1 :goal (explored cell23))
      :temporal-links (exp0 move)
```

```
  )
)


(:action patrol_agv_2
  :parameters (?r - agv)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell23))
  :effect (and (explored cell23) (explored cell13))
  :side-effect (and (at ?r cell13))
  :methods(

      :method patrol_agv_2_m
      :actions (exp0 (explore ?r cell23))
        (move (move-agv ?r cell23 cell13))
        (exp1 (explore ?r cell13))

      :precondition
      :causal-links (:init exp0 (at ?r cell23))
        (:init move (at ?r cell23))
        (move exp1 (at ?r cell13))
        (move :goal (at ?r cell13))
        (exp0 :goal (explored cell23))
        (exp1 :goal (explored cell13))
      :temporal-links (exp0 move)

  )
)


(:action patrol_agv_3
  :parameters (?r - agv)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell33))
  :effect (and (explored cell33) (explored cell43))
  :side-effect (and (at ?r cell43))
  :methods(

      :method patrol_agv_3_m
      :actions (exp0 (explore ?r cell33))
        (move (move-agv ?r cell33 cell43))
        (exp1 (explore ?r cell43))

      :precondition
      :causal-links (:init exp0 (at ?r cell33))
        (:init move (at ?r cell33))
        (move exp1 (at ?r cell43))
        (move :goal (at ?r cell43))
        (exp0 :goal (explored cell33))
        (exp1 :goal (explored cell43))
      :temporal-links (exp0 move)

  )
)
```

```
(:action patrol_agv_4
  :parameters (?r - agv)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell43))
  :effect (and (explored cell43) (explored cell33))
  :side-effect (and (at ?r cell33))
  :methods(

      :method patrol_agv_4_m
      :actions (exp0 (explore ?r cell43))
        (move (move-agv ?r cell43 cell33))
        (exp1 (explore ?r cell33))

      :precondition
      :causal-links (:init exp0 (at ?r cell43))
        (:init move (at ?r cell43))
        (move exp1 (at ?r cell33))
        (move :goal (at ?r cell33))
        (exp0 :goal (explored cell43))
        (exp1 :goal (explored cell33))
      :temporal-links (exp0 move)

  )
)


(:action patrol_aav_5
  :parameters (?r - aav)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell43))
  :effect (and (explored cell43) (explored cell43))
  :side-effect (and (at ?r cell43))
  :methods(

      :method patrol_aav_5_m
      :actions (exp (explore ?r cell43))

      :precondition
      :causal-links (:init exp (at ?r cell43))
        (:init :goal (at ?r cell43))
        (exp :goal (explored cell43))
      :temporal-links

  )
)


(:action patrol_agv_5
  :parameters (?r - agv)
  :agent (?r)
  :conflict-with (at ?r *)

  :precondition (and (at ?r cell41))
  :effect (and (explored cell41) (explored cell41))
```

```
  :side-effect (and (at ?r cell41))
  :methods(

      :method patrol_agv_5_m
      :actions (exp (explore ?r cell41))

      :precondition
      :causal-links (:init exp (at ?r cell41))
        (:init :goal (at ?r cell41))
        (exp :goal (explored cell41))
      :temporal-links

  )
)

)
```

Description du problème

```
(define (problem ex-repair-init)
  (:domain ex-repair)

  (:objects
    aav1 - aav
    agv1 - agv
    cell11 cell12 cell13 cell21 cell22 cell23 cell31 cell32 cell33 cell41
        cell42 cell43  - loc
  )

  (:init
    (= (distance-aav cell11 cell12) 1) (= (distance-agv cell11 cell12) 1)
    (= (distance-aav cell11 cell13) 2) (= (distance-agv cell11 cell13) 2)
    (= (distance-aav cell11 cell21) 1) (= (distance-agv cell11 cell21) 1)
    (= (distance-aav cell11 cell22) 2) (= (distance-agv cell11 cell22) 2)
    (= (distance-aav cell11 cell23) 3) (= (distance-agv cell11 cell23) 3)
    (= (distance-aav cell11 cell31) 2) (= (distance-agv cell11 cell31) 2)
    (= (distance-aav cell11 cell32) 3) (= (distance-agv cell11 cell32) 3)
    (= (distance-aav cell11 cell33) 4) (= (distance-agv cell11 cell33) 4)
    (= (distance-aav cell11 cell41) 3) (= (distance-agv cell11 cell41) 3)
    (= (distance-aav cell11 cell42) 4) (= (distance-agv cell11 cell42) 4)
    (= (distance-aav cell11 cell43) 5) (= (distance-agv cell11 cell43) 5)
    (= (distance-aav cell12 cell11) 1) (= (distance-agv cell12 cell11) 1)
    (= (distance-aav cell12 cell13) 1) (= (distance-agv cell12 cell13) 1)
    (= (distance-aav cell12 cell21) 2) (= (distance-agv cell12 cell21) 2)
    (= (distance-aav cell12 cell22) 1) (= (distance-agv cell12 cell22) 1)
    (= (distance-aav cell12 cell23) 2) (= (distance-agv cell12 cell23) 2)
    (= (distance-aav cell12 cell31) 3) (= (distance-agv cell12 cell31) 3)
    (= (distance-aav cell12 cell32) 2) (= (distance-agv cell12 cell32) 2)
    (= (distance-aav cell12 cell33) 3) (= (distance-agv cell12 cell33) 3)
    (= (distance-aav cell12 cell41) 4) (= (distance-agv cell12 cell41) 4)
    (= (distance-aav cell12 cell42) 3) (= (distance-agv cell12 cell42) 3)
    (= (distance-aav cell12 cell43) 4) (= (distance-agv cell12 cell43) 4)
    (= (distance-aav cell13 cell11) 2) (= (distance-agv cell13 cell11) 2)
    (= (distance-aav cell13 cell12) 1) (= (distance-agv cell13 cell12) 1)
    (= (distance-aav cell13 cell21) 3) (= (distance-agv cell13 cell21) 3)
    (= (distance-aav cell13 cell22) 2) (= (distance-agv cell13 cell22) 2)
    (= (distance-aav cell13 cell23) 1) (= (distance-agv cell13 cell23) 1)
    (= (distance-aav cell13 cell31) 4) (= (distance-agv cell13 cell31) 4)
    (= (distance-aav cell13 cell32) 3) (= (distance-agv cell13 cell32) 3)
    (= (distance-aav cell13 cell33) 2) (= (distance-agv cell13 cell33) 2)
```

```
41      (= (distance-aav cell13 cell41) 5) (= (distance-agv cell13 cell41) 5)
42      (= (distance-aav cell13 cell42) 4) (= (distance-agv cell13 cell42) 4)
43      (= (distance-aav cell13 cell43) 3) (= (distance-agv cell13 cell43) 3)
44      (= (distance-aav cell21 cell11) 1) (= (distance-agv cell21 cell11) 1)
45      (= (distance-aav cell21 cell12) 2) (= (distance-agv cell21 cell12) 2)
46      (= (distance-aav cell21 cell13) 3) (= (distance-agv cell21 cell13) 3)
47      (= (distance-aav cell21 cell22) 1) (= (distance-agv cell21 cell22) 1)
48      (= (distance-aav cell21 cell23) 2) (= (distance-agv cell21 cell23) 2)
49      (= (distance-aav cell21 cell31) 1) (= (distance-agv cell21 cell31) 1)
50      (= (distance-aav cell21 cell32) 2) (= (distance-agv cell21 cell32) 2)
51      (= (distance-aav cell21 cell33) 3) (= (distance-agv cell21 cell33) 3)
52      (= (distance-aav cell21 cell41) 2) (= (distance-agv cell21 cell41) 2)
53      (= (distance-aav cell21 cell42) 3) (= (distance-agv cell21 cell42) 3)
54      (= (distance-aav cell21 cell43) 4) (= (distance-agv cell21 cell43) 4)
55      (= (distance-aav cell22 cell11) 2) (= (distance-agv cell22 cell11) 2)
56      (= (distance-aav cell22 cell12) 1) (= (distance-agv cell22 cell12) 1)
57      (= (distance-aav cell22 cell13) 2) (= (distance-agv cell22 cell13) 2)
58      (= (distance-aav cell22 cell21) 1) (= (distance-agv cell22 cell21) 1)
59      (= (distance-aav cell22 cell23) 1) (= (distance-agv cell22 cell23) 1)
60      (= (distance-aav cell22 cell31) 2) (= (distance-agv cell22 cell31) 2)
61      (= (distance-aav cell22 cell32) 1) (= (distance-agv cell22 cell32) 1)
62      (= (distance-aav cell22 cell33) 2) (= (distance-agv cell22 cell33) 2)
63      (= (distance-aav cell22 cell41) 3) (= (distance-agv cell22 cell41) 3)
64      (= (distance-aav cell22 cell42) 2) (= (distance-agv cell22 cell42) 2)
65      (= (distance-aav cell22 cell43) 3) (= (distance-agv cell22 cell43) 3)
66      (= (distance-aav cell23 cell11) 3) (= (distance-agv cell23 cell11) 3)
67      (= (distance-aav cell23 cell12) 2) (= (distance-agv cell23 cell12) 2)
68      (= (distance-aav cell23 cell13) 1) (= (distance-agv cell23 cell13) 1)
69      (= (distance-aav cell23 cell21) 2) (= (distance-agv cell23 cell21) 2)
70      (= (distance-aav cell23 cell22) 1) (= (distance-agv cell23 cell22) 1)
71      (= (distance-aav cell23 cell31) 3) (= (distance-agv cell23 cell31) 3)
72      (= (distance-aav cell23 cell32) 2) (= (distance-agv cell23 cell32) 2)
73      (= (distance-aav cell23 cell33) 1) (= (distance-agv cell23 cell33) 1)
74      (= (distance-aav cell23 cell41) 4) (= (distance-agv cell23 cell41) 4)
75      (= (distance-aav cell23 cell42) 3) (= (distance-agv cell23 cell42) 3)
76      (= (distance-aav cell23 cell43) 2) (= (distance-agv cell23 cell43) 2)
77      (= (distance-aav cell31 cell11) 2) (= (distance-agv cell31 cell11) 2)
78      (= (distance-aav cell31 cell12) 3) (= (distance-agv cell31 cell12) 3)
79      (= (distance-aav cell31 cell13) 4) (= (distance-agv cell31 cell13) 4)
80      (= (distance-aav cell31 cell21) 1) (= (distance-agv cell31 cell21) 1)
81      (= (distance-aav cell31 cell22) 2) (= (distance-agv cell31 cell22) 2)
82      (= (distance-aav cell31 cell23) 3) (= (distance-agv cell31 cell23) 3)
83      (= (distance-aav cell31 cell32) 1) (= (distance-agv cell31 cell32) 1)
84      (= (distance-aav cell31 cell33) 2) (= (distance-agv cell31 cell33) 2)
85      (= (distance-aav cell31 cell41) 1) (= (distance-agv cell31 cell41) 1)
86      (= (distance-aav cell31 cell42) 2) (= (distance-agv cell31 cell42) 2)
87      (= (distance-aav cell31 cell43) 3) (= (distance-agv cell31 cell43) 3)
88      (= (distance-aav cell32 cell11) 3) (= (distance-agv cell32 cell11) 3)
89      (= (distance-aav cell32 cell12) 2) (= (distance-agv cell32 cell12) 2)
90      (= (distance-aav cell32 cell13) 3) (= (distance-agv cell32 cell13) 3)
91      (= (distance-aav cell32 cell21) 2) (= (distance-agv cell32 cell21) 2)
92      (= (distance-aav cell32 cell22) 1) (= (distance-agv cell32 cell22) 1)
93      (= (distance-aav cell32 cell23) 2) (= (distance-agv cell32 cell23) 2)
94      (= (distance-aav cell32 cell31) 1) (= (distance-agv cell32 cell31) 1)
95      (= (distance-aav cell32 cell33) 1) (= (distance-agv cell32 cell33) 1)
96      (= (distance-aav cell32 cell41) 2) (= (distance-agv cell32 cell41) 2)
97      (= (distance-aav cell32 cell42) 1) (= (distance-agv cell32 cell42) 1)
98      (= (distance-aav cell32 cell43) 2) (= (distance-agv cell32 cell43) 2)
99      (= (distance-aav cell33 cell11) 4) (= (distance-agv cell33 cell11) 4)
```

```
100     (= (distance-aav cell33 cell12) 3) (= (distance-agv cell33 cell12) 3)
101     (= (distance-aav cell33 cell13) 2) (= (distance-agv cell33 cell13) 2)
102     (= (distance-aav cell33 cell21) 3) (= (distance-agv cell33 cell21) 3)
103     (= (distance-aav cell33 cell22) 2) (= (distance-agv cell33 cell22) 2)
104     (= (distance-aav cell33 cell23) 1) (= (distance-agv cell33 cell23) 1)
105     (= (distance-aav cell33 cell31) 2) (= (distance-agv cell33 cell31) 2)
106     (= (distance-aav cell33 cell32) 1) (= (distance-agv cell33 cell32) 1)
107     (= (distance-aav cell33 cell41) 3) (= (distance-agv cell33 cell41) 3)
108     (= (distance-aav cell33 cell42) 2) (= (distance-agv cell33 cell42) 2)
109     (= (distance-aav cell33 cell43) 1) (= (distance-agv cell33 cell43) 1)
110     (= (distance-aav cell41 cell11) 3) (= (distance-agv cell41 cell11) 3)
111     (= (distance-aav cell41 cell12) 4) (= (distance-agv cell41 cell12) 4)
112     (= (distance-aav cell41 cell13) 5) (= (distance-agv cell41 cell13) 5)
113     (= (distance-aav cell41 cell21) 2) (= (distance-agv cell41 cell21) 2)
114     (= (distance-aav cell41 cell22) 3) (= (distance-agv cell41 cell22) 3)
115     (= (distance-aav cell41 cell23) 4) (= (distance-agv cell41 cell23) 4)
116     (= (distance-aav cell41 cell31) 1) (= (distance-agv cell41 cell31) 1)
117     (= (distance-aav cell41 cell32) 2) (= (distance-agv cell41 cell32) 2)
118     (= (distance-aav cell41 cell33) 3) (= (distance-agv cell41 cell33) 3)
119     (= (distance-aav cell41 cell42) 1) (= (distance-agv cell41 cell42) 1)
120     (= (distance-aav cell41 cell43) 2) (= (distance-agv cell41 cell43) 2)
121     (= (distance-aav cell42 cell11) 4) (= (distance-agv cell42 cell11) 4)
122     (= (distance-aav cell42 cell12) 3) (= (distance-agv cell42 cell12) 3)
123     (= (distance-aav cell42 cell13) 4) (= (distance-agv cell42 cell13) 4)
124     (= (distance-aav cell42 cell21) 3) (= (distance-agv cell42 cell21) 3)
125     (= (distance-aav cell42 cell22) 2) (= (distance-agv cell42 cell22) 2)
126     (= (distance-aav cell42 cell23) 3) (= (distance-agv cell42 cell23) 3)
127     (= (distance-aav cell42 cell31) 2) (= (distance-agv cell42 cell31) 2)
128     (= (distance-aav cell42 cell32) 1) (= (distance-agv cell42 cell32) 1)
129     (= (distance-aav cell42 cell33) 2) (= (distance-agv cell42 cell33) 2)
130     (= (distance-aav cell42 cell41) 1) (= (distance-agv cell42 cell41) 1)
131     (= (distance-aav cell42 cell43) 1) (= (distance-agv cell42 cell43) 1)
132     (= (distance-aav cell43 cell11) 5) (= (distance-agv cell43 cell11) 5)
133     (= (distance-aav cell43 cell12) 4) (= (distance-agv cell43 cell12) 4)
134     (= (distance-aav cell43 cell13) 3) (= (distance-agv cell43 cell13) 3)
135     (= (distance-aav cell43 cell21) 4) (= (distance-agv cell43 cell21) 4)
136     (= (distance-aav cell43 cell22) 3) (= (distance-agv cell43 cell22) 3)
137     (= (distance-aav cell43 cell23) 2) (= (distance-agv cell43 cell23) 2)
138     (= (distance-aav cell43 cell31) 3) (= (distance-agv cell43 cell31) 3)
139     (= (distance-aav cell43 cell32) 2) (= (distance-agv cell43 cell32) 2)
140     (= (distance-aav cell43 cell33) 1) (= (distance-agv cell43 cell33) 1)
141     (= (distance-aav cell43 cell41) 2) (= (distance-agv cell43 cell41) 2)
142     (= (distance-aav cell43 cell42) 1) (= (distance-agv cell43 cell42) 1)
143
144
145     (aav-allowed cell11) (aav-allowed cell21) (aav-allowed cell31) (aav-
            allowed cell41)
146     (aav-allowed cell12) (aav-allowed cell22) (aav-allowed cell32) (aav-
            allowed cell42)
147     (aav-allowed cell43)
148
149     (agv-allowed cell41)
150     (agv-allowed cell12) (agv-allowed cell22) (agv-allowed cell32) (agv-
            allowed cell42)
151     (agv-allowed cell13) (agv-allowed cell23) (agv-allowed cell33) (agv-
            allowed cell43)
152
153
154     (at aav1 cell12)
```

```
155     (at agv1 cell12)
156   )
157
158   (:goal (and
159     (explored cell11) (explored cell21) (explored cell31) (explored cell41)
160     (explored cell13) (explored cell23) (explored cell33) (explored cell43)
161   ))
162 )
```

E

Résultats des scénarios en conditions réelles

Ces descriptions sont issues du rapport final du projet ACTION concernant les scénarios aéroterrestres (rapport LT0.5, Novembre 2015).

****

Neuf missions ont été lancées les 19 et 20 octobre 2015 : 4 entre 15h et 17h30 le lundi après-midi et 5 entre 10h30 et 15h le mardi. Un suivi de cible a aussi été ponctuellement réalisé par le ReSSAC1 à une altitude de 10m au lieu de 30m, sans définir de mission.

Le scénario nominal a été validé au huitième lancement de mission suite à l'occurrence de défauts matériels et logiciels non désirés. Les missions nommées « Mission VIi » effectivement jouées sont décrites ci-dessous.

Pour chaque mission sont donnés la timeline finale qui montre l'état des véhicules à l'arrêt de la mission et les itinéraires réalisés par les véhicules réels et simulés dans 4 figures : tous les véhicules, les AAV ReSSAC de l'ONERA, les AGV du LAAS et les AGV de DGA. Les itinéraires pour les AAV comprennent le décollage et l'atterrissage manuels. Pour ces figures sur les itinéraires, on trouvera en haut à gauche tous les véhicules, en haut à droite les drones aériens ONERA, en bas à gauche les robots terrestres LAAS, en bas à droite les robots terrestres DGA.

Mission VI.1

Début de patrouille puis trois problèmes techniques décrits ci-dessous avec les solutions adoptées pour la mission suivante. Cinq véhicules sont déclarés morts dans cette mission. Plusieurs réparations « Un AXV devient non opérationnel » sont déclenchées et les véhicules restants finissent la mission. Deux véhicules de rechange sont utilisés.

- La navigation autonome de Momo ne lui permet pas de pénétrer dans la zone herbée : il est échangé avec un Effibot auquel on enlève l'évitement d'obstacles ;
- La couche MONO des Effibot reçoit du bas niveau l'information que le point de passage n'est pas accessible : l'opérateur sécurité vérifie que la connaissance GPS était fausse en début suite à une mauvaise initialisation (connue mais oubliée dans l'action de la démonstration) ; avant chaque départ, les Effibot doivent faire un aller-retour pour recaler la position GPS dont le cap de départ ;
- Deux points de passage pour un ReSSAC sont trop proches (moins de 15m) : le drone déclenche un comportement de changement d'altitude, prévu lorsque deux

points se supperposent ; ce comportement, qui survient pour la première fois en essai (fortement dépendant des conditions d'arrivée à un point de passage, elles-mêmes fortement dépendantes du vent), a été par la suite désactivé.

L'analyse de la timeline de la figure E.1 et des itinéraires de la figure E.2 montre les résultats suivants :

- Durée de la mission ∼9mn ;
- Un vent fort pour le ReSSAC2 ;
- Des variations sur les points d'attente des ReSSAC réels aux rendez-vous ;
- ReSSAC1 finit sa patrouille ;
- ReSSAC2 meurt mais n'est pas remplacé (pas de rechange pour le deuxième drone aérien, ReSSAC3 est le drone de rechange de ReSSAC1) ;
- Mana et Minnie finissent leur patrouille ;
- Momo, mort mais ayant déjà observé ses points dans sa patrouille, n'est pas remplacé ;
- Les Effibot réel n'ont pas parcouru beaucoup de chemin avant d'être remplacés : Effibot1, Effibot2 et Effibot3, morts, sont remplacés par Effibot4 et Effibot5 qui finissent leur patrouille ; la reprise par Effibot4 simulé des actions de Effibot1 et de Effibot2 lui fait traverser tout le terrain ;
- Mona et ReSSAC3, véhicules de rechange, ne sont pas utilisés
- Aucun rendez-vous n'est réalisé.

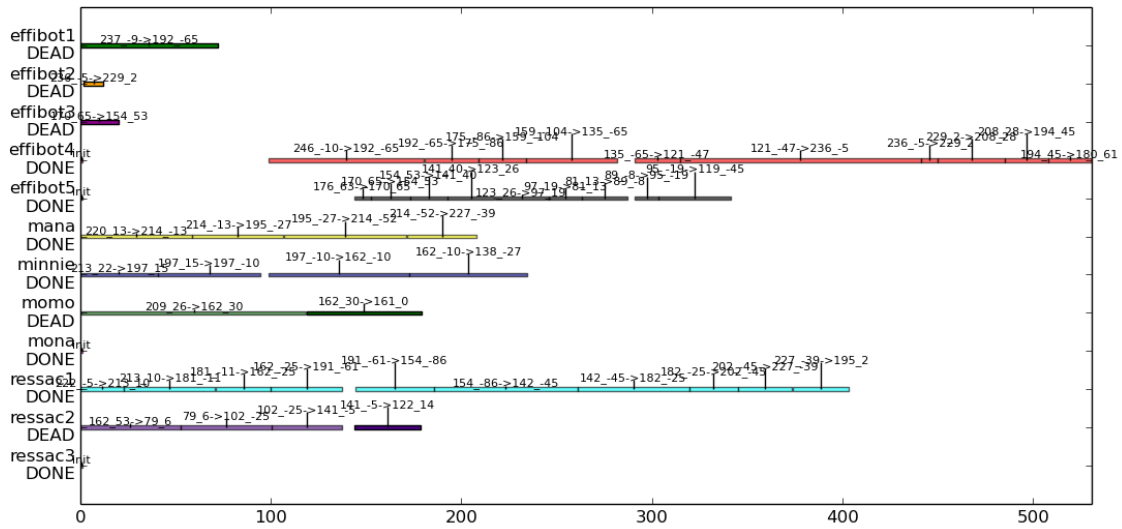


Figure E.1 – Timeline de la mission jouée VI.1.

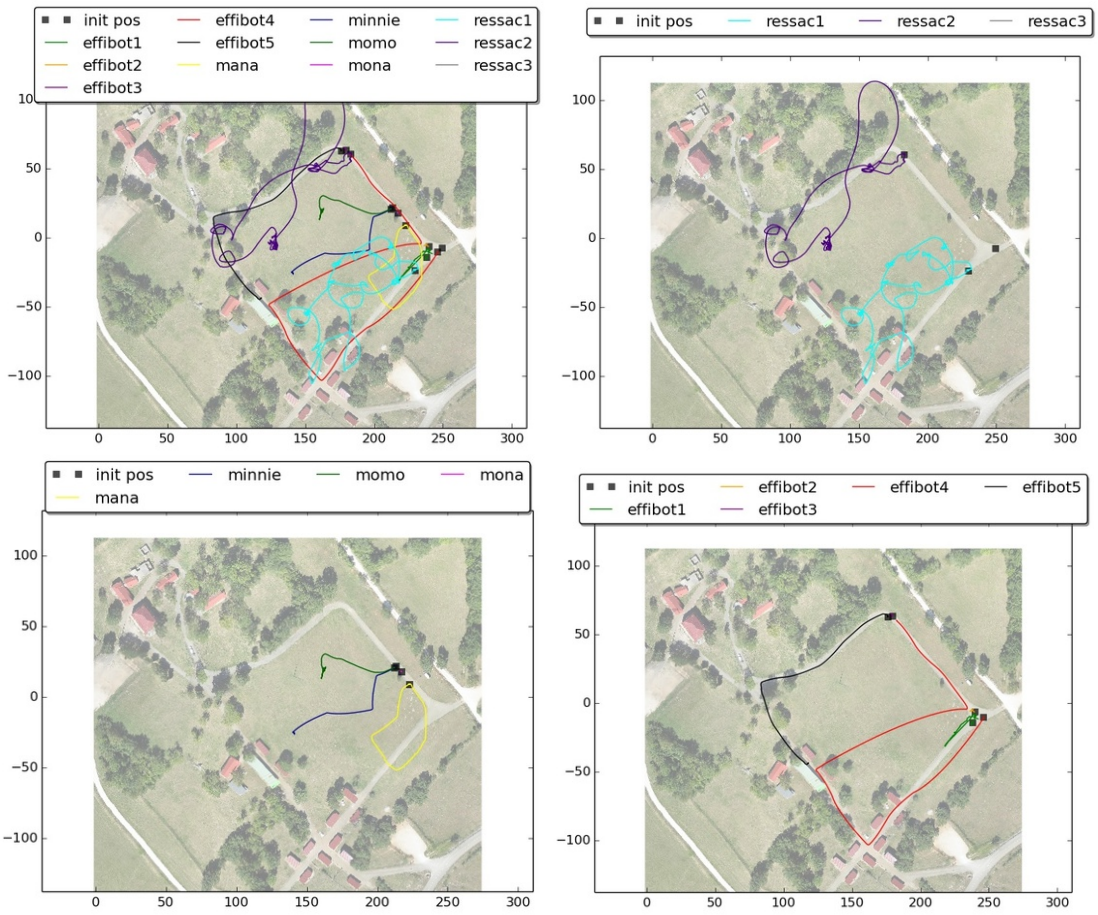


Figure E.2 – Itinéraires de la mission jouée VI.1 (véhicules réels et simulés).

Mission VI.2

Début de patrouille, puis deux aléas techniques pour lesquels les véhicules sont déclarés morts (« non opérationnel ») : Momo n'arrive pas à avancer sur la route dans la zone du nord-est suite à un problème sur son Velodyne ; Effibot1 s'arrête sans connaissance de la raison. Le déplacement sinueux de certains robots terrestres déclenche plusieurs réparations du plan suite à leur absence aux rendez-vous (aléas « Un AXV est en retard à un rendez-vous »).

L'analyse de la timeline de la figure E.3 et des itinéraires de la figure E.4 montre les résultats suivants :

- Durée de la mission ∼12mn ;
- ReSSAC1 et ReSSAC2 finissent leur patrouille ;
- Mana et Minnie finissent leur patrouille ;
- Momo, mort, (il a été déclaré comme se déplaçant sur le chemin), est remplacé par Effibot5 qui finit sa patrouille ;
- Effibot1, mort, est remplacé par Effibot4 qui finit sa patrouille ;
- Effibot2 et Effibot3 (Effibot3 a été déclaré comme se déplaçant sur l'herbe) finissent leur patrouille ;
- Mona et ReSSAC3, véhicules de rechange, ne sont pas utilisés ;
- Un rendez-vous a lieu entre Effibot3 et Minnie.

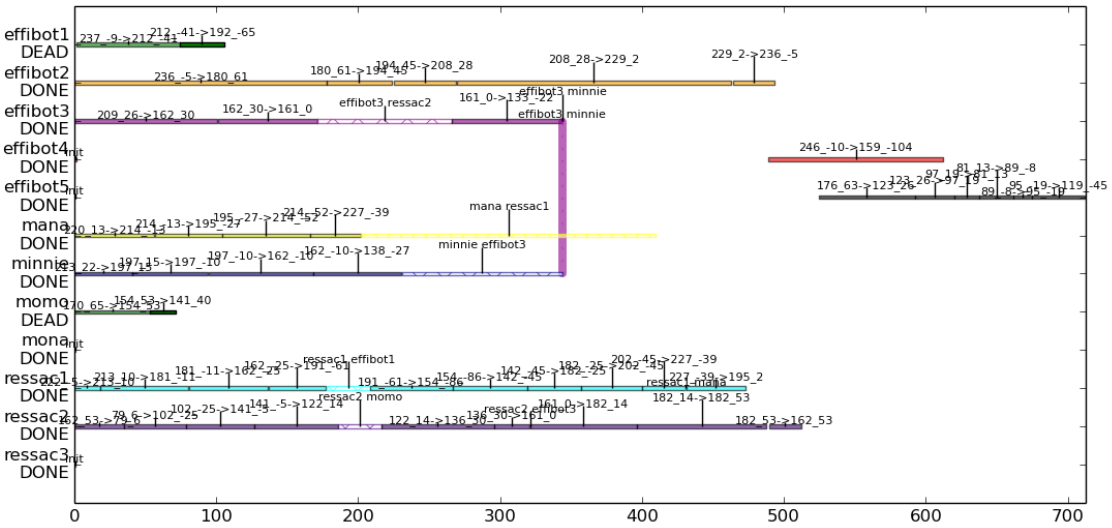


Figure E.3 – Timeline de la mission jouée VI.2.

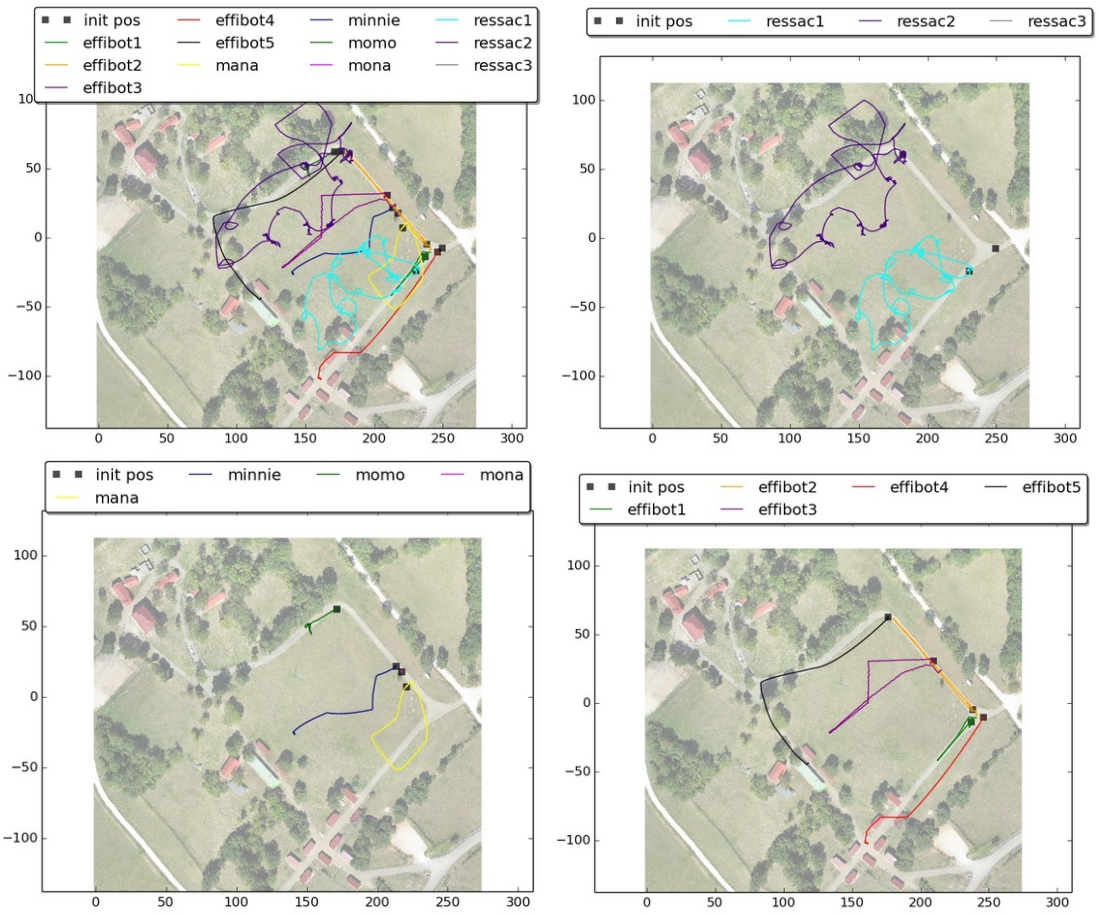


Figure E.4 – Itinéraires de la mission jouée VI.2 (véhicules réels et simulés).

Mission VI.3

Début de patrouille puis un aléa technique avec Momo qui est toujours bloqué sur la route dans la zone du nord-est et semble être perdu. Une réparation dans laquelle il est indiqué non opérationnel.

L'affichage des états sur la timeline de la figure E.5 est étrange. La mission est finie quand tous les véhicules ont fini et passent donc dans l'état *Done* ; sinon ils sont susceptibles d'aider à la réparation d'un plan (comme véhicule de rechange). On ne devrait donc pas avoir à la fois les états *Running* et les états *Done* : il y a peut-être eu un problème de communication qui a validé la fin de la mission pour certains véhicules mais pas pour d'autres. Effibot1 est le seul a ne pas avoir fini ses actions, dont son rendez-vous. L'analyse de cette figure et des itinéraires de la figure E.6 montre les autres résultats suivants :

- Durée de la mission ∼10mn ;
- ReSSAC1 et ReSSAC2 finissent leur patrouille ;
- Minnie finit sa patrouille, Mana aussi (il a fait son rendez-vous) ;
- Momo, mort, (il a été déclaré comme se déplaçant sur le chemin), a fini d'observer ses points et n'est donc pas remplacé ;
- Effibot1 est déclaré mort (mais il n'a pas proprement informé la visualisation qui ne le sait pas), ses actions sont reprises par Effibot4 ; Effibot2 et Effibot3 les ont finies ;
- Cinq rendez-vous des six rendez-vous prévus ont eu lieu.

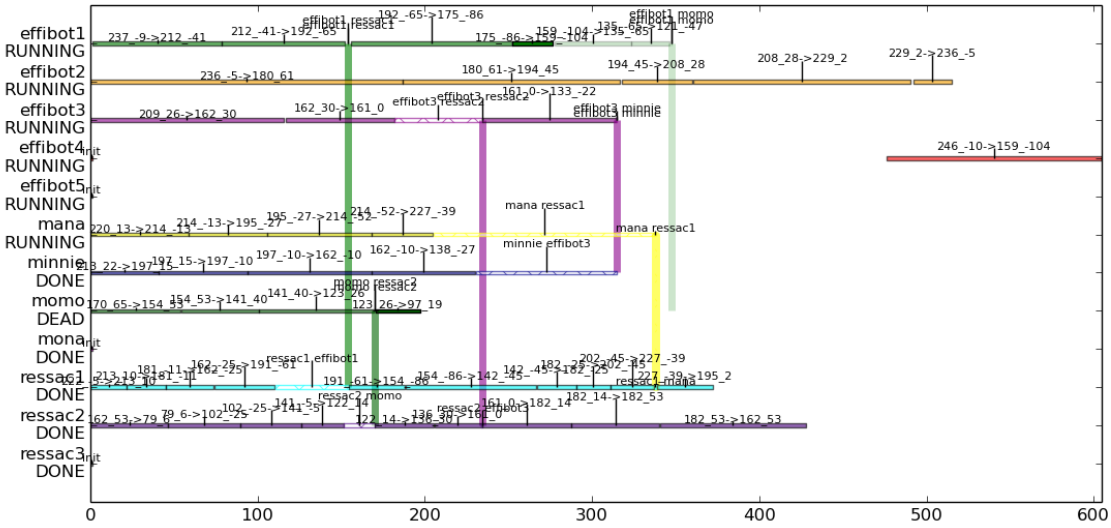


Figure E.5 – Timeline de la mission jouée VI.3.

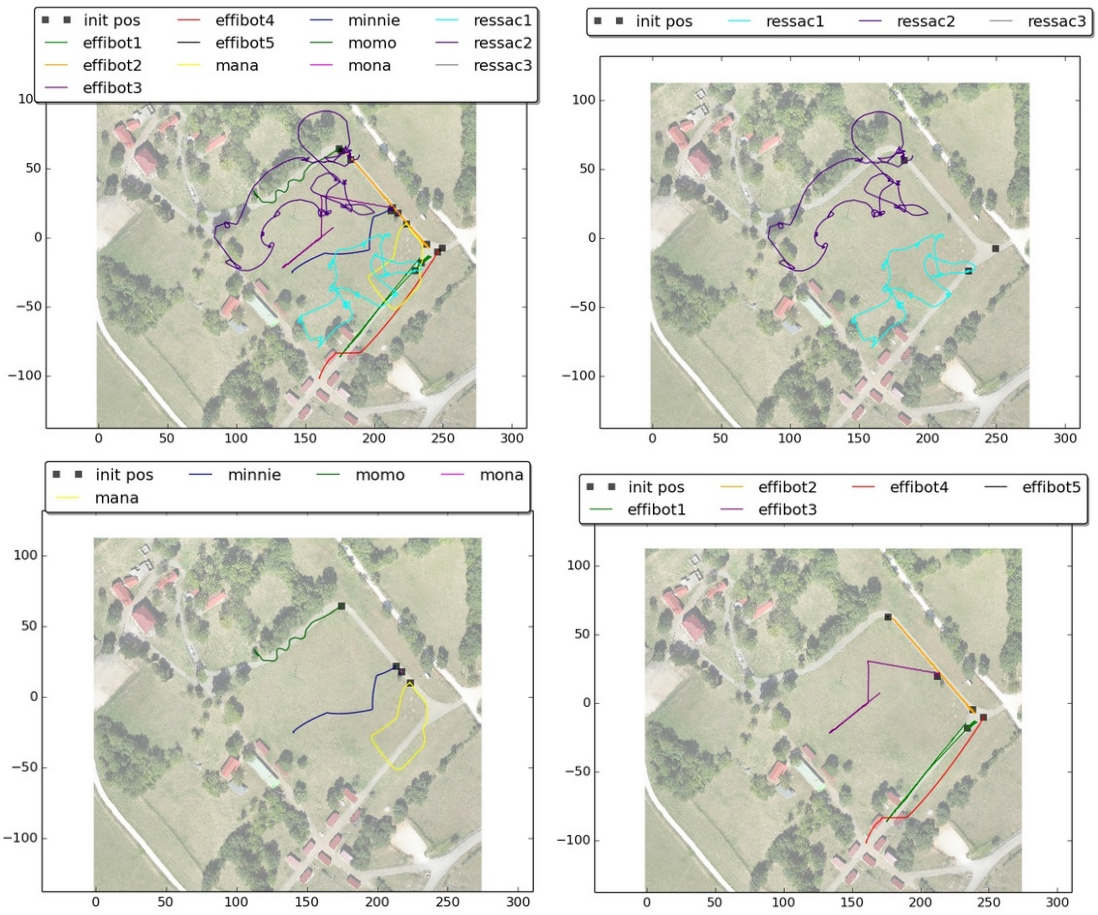


Figure E.6 – Itinéraires de la mission jouée VI.3 (véhicules réels et simulés).

Mission VI.4

Début de patrouille, puis une cible est positionnée sur le chemin de Mana. Momo (qui a remplacé Effibot3 depuis la mission VI.1) a des problèmes de navigation sur sa fin de parcours avant son rendez-vous avec Effibot3, il a sans doute détecté un obstacle. Mana détecte la cible et la suit correctement. ReSSAC1 est prévenu mais ne confirme pas la réception de l'information de suivi de cible : il est déclaré hors service et repris en main par le pilote de sécurité pour l'atterrissage. Ses actions de patrouille sont récupérées par le drone de rechange ReSSAC3. L'opérateur déclare le suivi de cible suffisant et Mana réintègre l'équipe de véhicules, en tant que robot de rechange puisque ses actions sont déjà affectées et qu'elles se déroulent nominalement jusqu'à la fin. La mission n'est pas considérée finie par les véhicules (en état *Running*) car Momo n'a pas été déclaré mort.

L'analyse de la timeline de la figure E.7 et des itinéraires de la figure E.8 montre les résultats suivants :

- Durée de la mission ∼13mn ;
- ReSSAC1 est déclaré mort et remplacé par ReSSAC3 ; ReSSAC2 finit sa patrouille ;
- Mana détecte et suit une cible ; ses actions sont reprises par le robot terrestre de rechange Mona ; Minnie finit sa patrouille ;
- Effibot1 est mort mais après l'observation de ses points, il n'est pas remplacé ;
- Effibot2 et Effibot3 finissent leur patrouille.

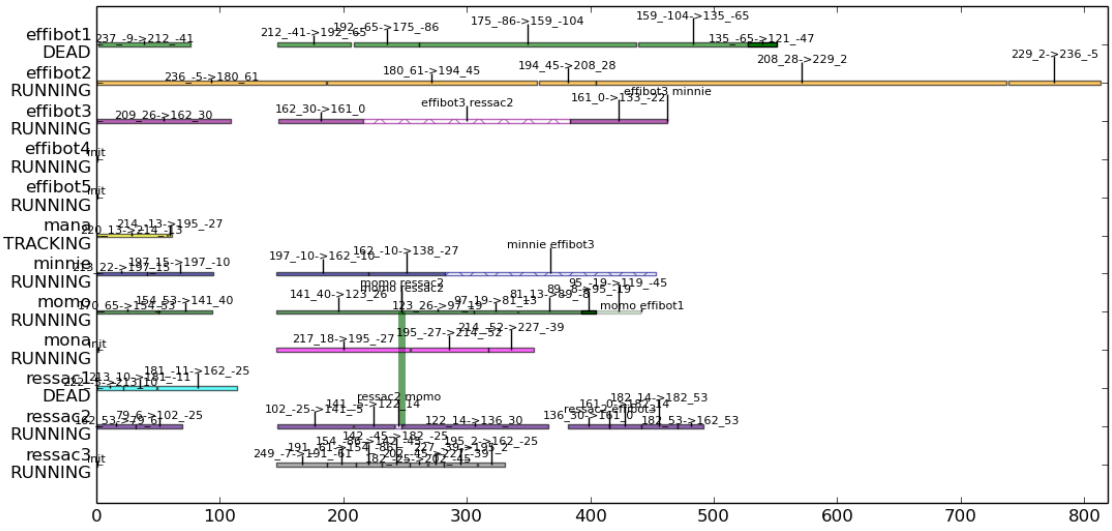


Figure E.7 – Timeline de la mission jouée VI.4.

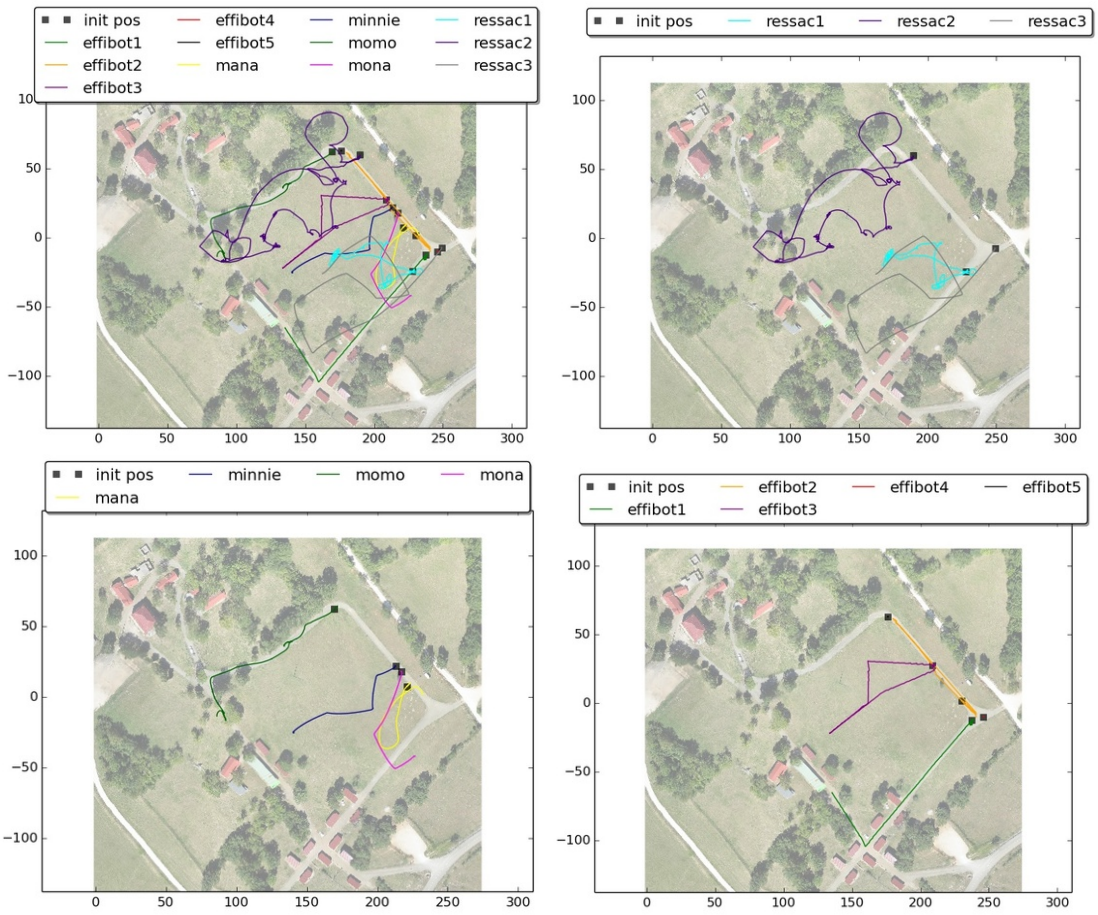


Figure E.8 – Itinéraires de la mission jouée VI.4 (véhicules réels et simulés).

Mission VI.5

Patrouille avec Momo simulé car celui-ci ne reçoit pas de données GPS (le lendemain matin). Une cible est mise sur la zone, mais son traceur n'est pas assez chargé et ReSSAC1 n'arrive pas à la suivre. Mana détecte la cible et la suit. Les autres véhicules finissent leur patrouille.

L'analyse de la timeline de la figure E.9 et des itinéraires de la figure E.10 montre les résultats suivants :

- Durée de la mission ∼9mn ;
- Mana détecte et suit une cible ; ces actions sont reprises par le robot terrestre simulé Mona ;
- ReSSAC1, informé sur la cible, la suit également ; ces actions sont reprises par le drone aérien simulé ReSSAC3 ;
- ReSSAC2 finit sa patrouille ;
- Effibot1 et Effibot2 n'ont pas fini leurs actions, Effibot3 oui ;
- Un rendez-vous a eu lieu.

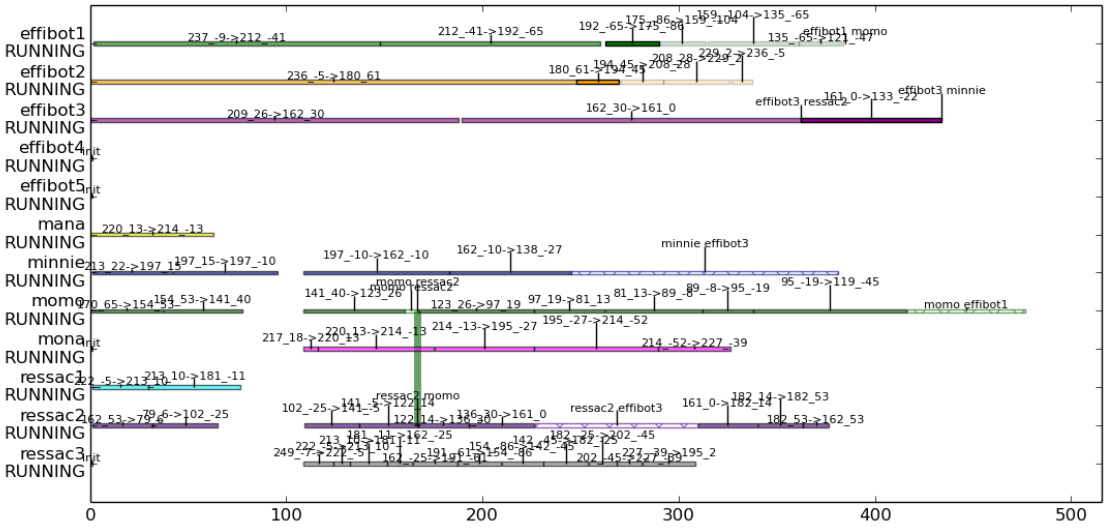


Figure E.9 – Timeline de la mission jouée VI.5.

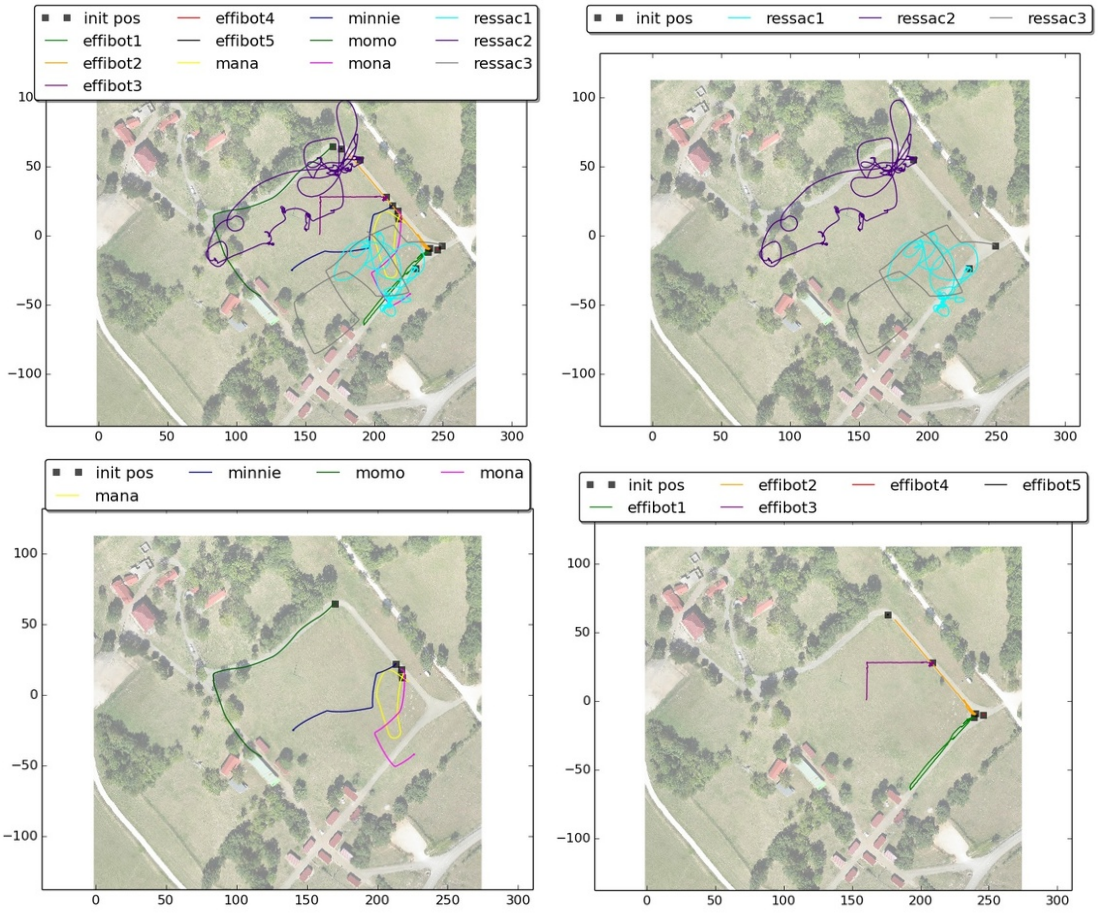


Figure E.10 – Itinéraires de la mission jouée VI.5 (véhicules réels et simulés).

Mission VI.6

Choix de simuler les drones aériens pour valider les robots terrestres présentant beaucoup d'aléas techniques depuis le début de la démonstration, mais les aléas continuent : Momo a un problème de localisation et est déclaré hors service et le plan réparé fait reprendre ses actions par Effibot1. Mana suit sa cible.

L'analyse de la timeline de la figure E.11 et des itinéraires de la figure E.12 montre les résultats suivants :

- Durée de la mission ∼10mn ;
- ReSSAC1 et ReSSAC2, simulés, finissent leur patrouille ;
- Mana détecte et suit une cible ; ces actions sont reprises par le robot terrestre simulé Mona ;
- Minnie finit sa patrouille ;
- Momo est déclaré mort et ses actions sont aussi reprises par Mona ;
- Effibot2 et Effibot3 finissent leur patrouille, pas Effibot1 qui a été repris en main et publie son retour à sa base ;
- Deux rendez-vous sont réalisés.

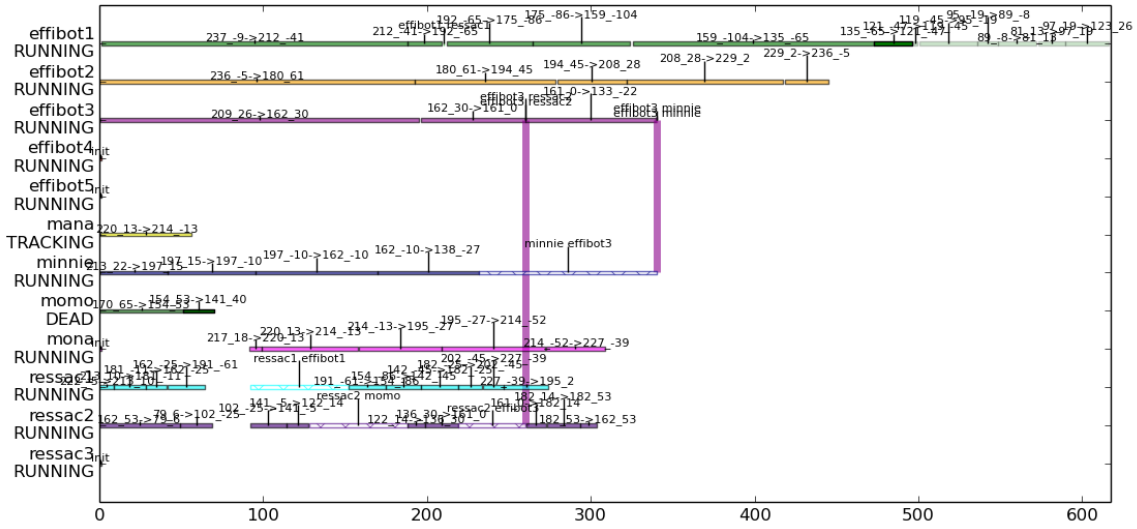


Figure E.11 – Timeline de la mission jouée VI.6.

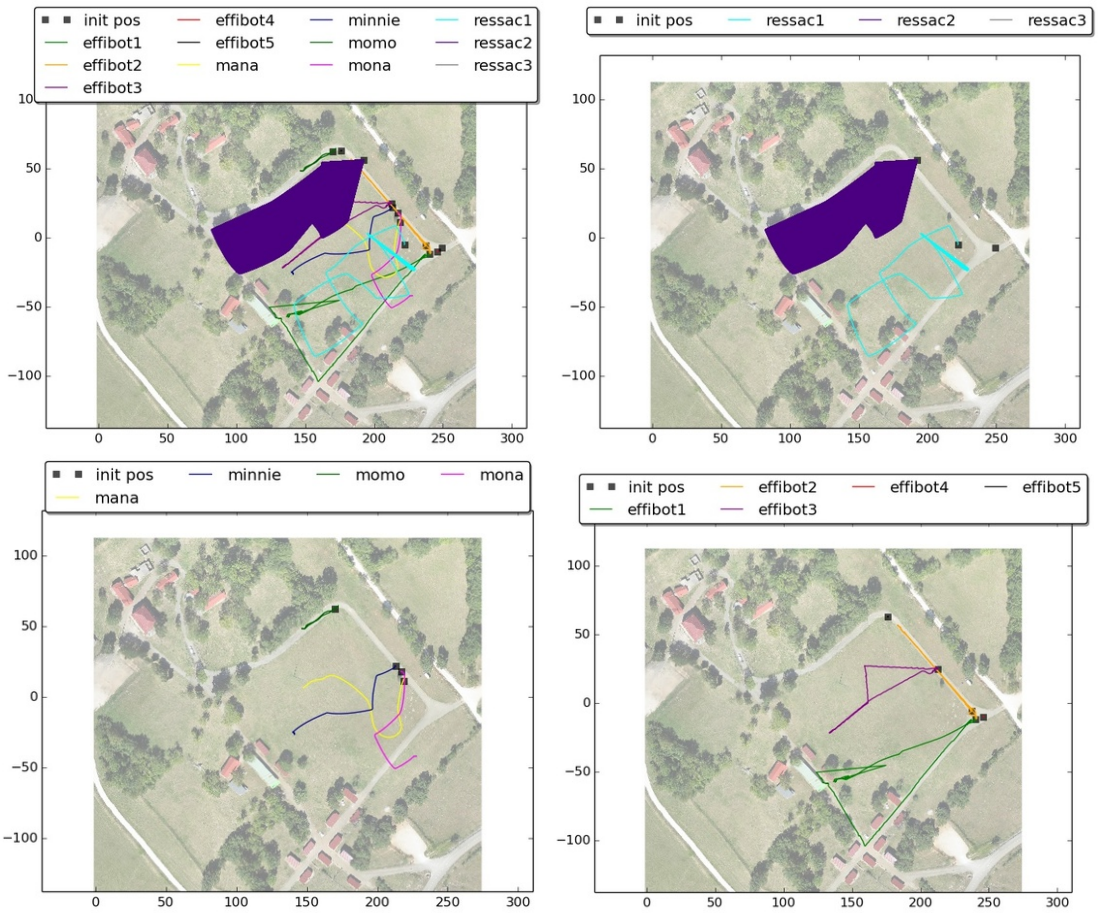


Figure E.12 – Itinéraires de la mission jouée VI.6 (véhicules réels et simulés). La zone violette est un artefact dû à la présence d'un ReSSAC2 réel en même temps que le ReSSAC2 simulé : deux positions sont émises en parallèle et donnent une impression de coloriage.

Mission VI.7

Les drones aériens sont simulés. Pour assurer le seul rendez-vous entre deux vrais robots terrestres qui n'a pas eu lieu depuis le début entre Effibot1 et Momo, Momo et Mana sont échangés. Momo se retrouve sur la zone herbée et sa navigation fonctionne. Effibot1 et Mana réalisent leur rendez-vous.

L'analyse des itinéraires de la figure E.14 et de la timeline de la figure E.13 montre les résultats suivants :

- Durée de la mission ∼14mn ;
- ReSSAC1 et ReSSAC2, simulés, finissent leur patrouille ;
- Mana, Minnie et Momo finissent leur patrouille ;
- Effibot1, Effibot2 et Effibot3 finissent leur patrouille ;
- Deux rendez-vous sont réalisés ;
- Aucun véhicule de rechange n'est utilisé ;
- Cinq rendez-vous sur six ont lieu ; celui entre Effibot1 et ReSSAC1 n'a pas eu lieu car ReSSAC1 ne l'a pas attendu (le premier déplacement d'Effibot1 était trop en retard).

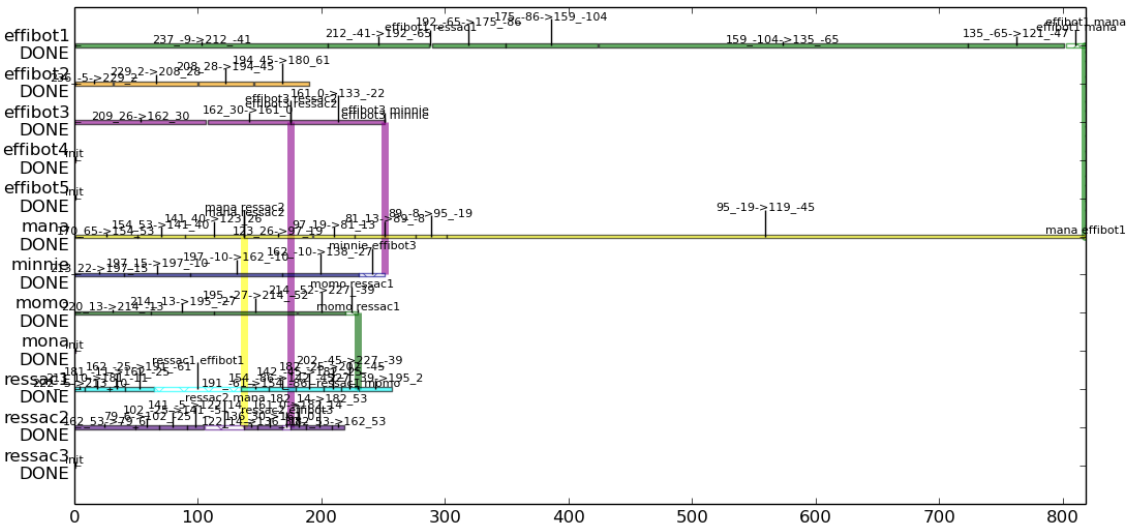


Figure E.13 – Timeline de la mission jouée VI.7.

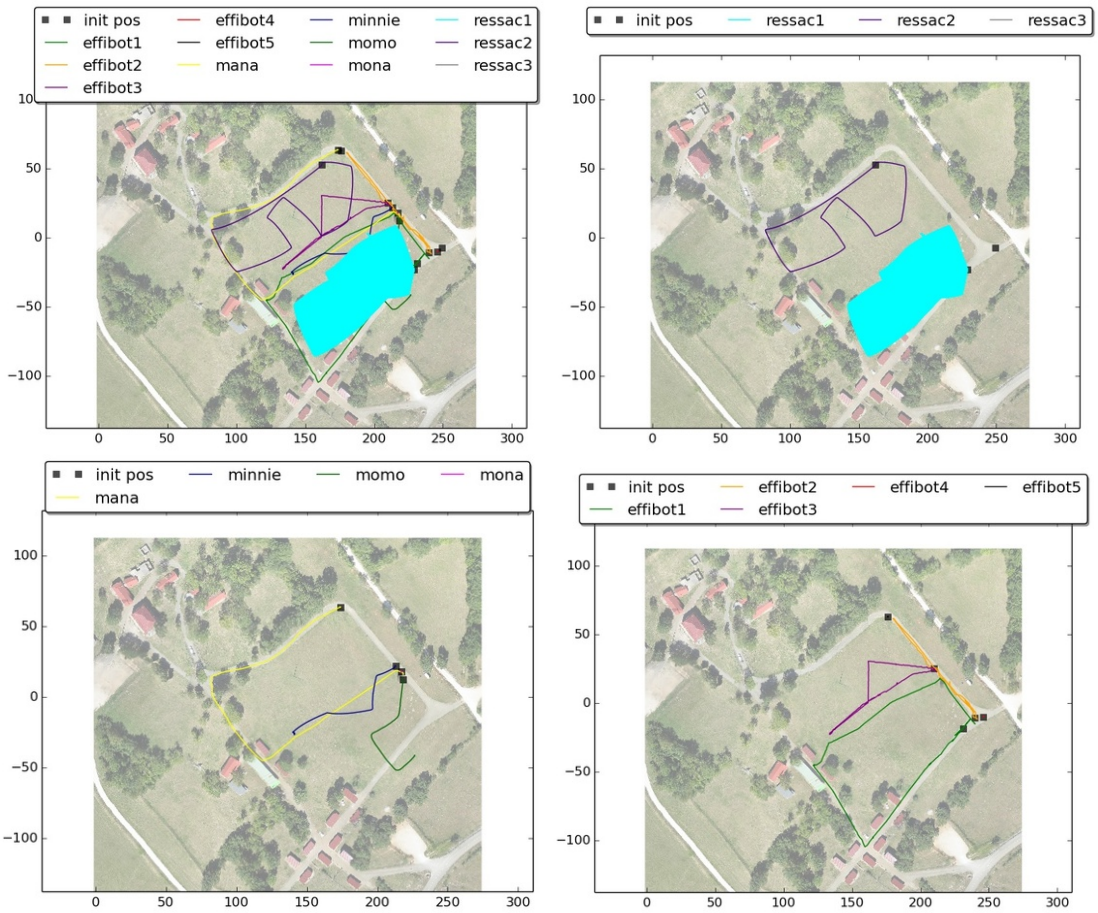


Figure E.14 – Itinéraires de la mission jouée VI.7 (véhicules réels et simulés). La zone bleue clair est un artefact dû à la présence d'un ReSSAC1 réel en même temps que le ReSSAC1 simulé : deux positions sont émises en parallèle et donnent une impression de coloriage.

Mission VI.8

La patrouille est relancée avec la configuration de la mission VI.7 (Mana à la place de Effibot1, Effibot1 à la place de Momo et Momo à la place de Mana sur le plan initial donné) et les drones aériens réels, sans cible : tous les véhicules font leur part de patrouille, aucune réparation n'est nécessaire (des décalages temporels dans la réalisation des actions n'impactent pas les rendez-vous) et tous ces rendez-vous sont réalisés. Le scénario VI.1 nominal est démontré avec les 7 véhicules réels, deux ReSSAC, Mana et Momo et les trois Effibot.

L'analyse de la timeline de la figure E.15 et des itinéraires de la figure E.16 montre les résultats suivants pour ce scénario nominal :

– Durée de la mission ∼8mn ;

– Tous les véhicules réels finissent leur patrouille ;

– Aucun véhicule de rechange n'est utilisé ;

– Les six rendez-vous prévus ont lieu.

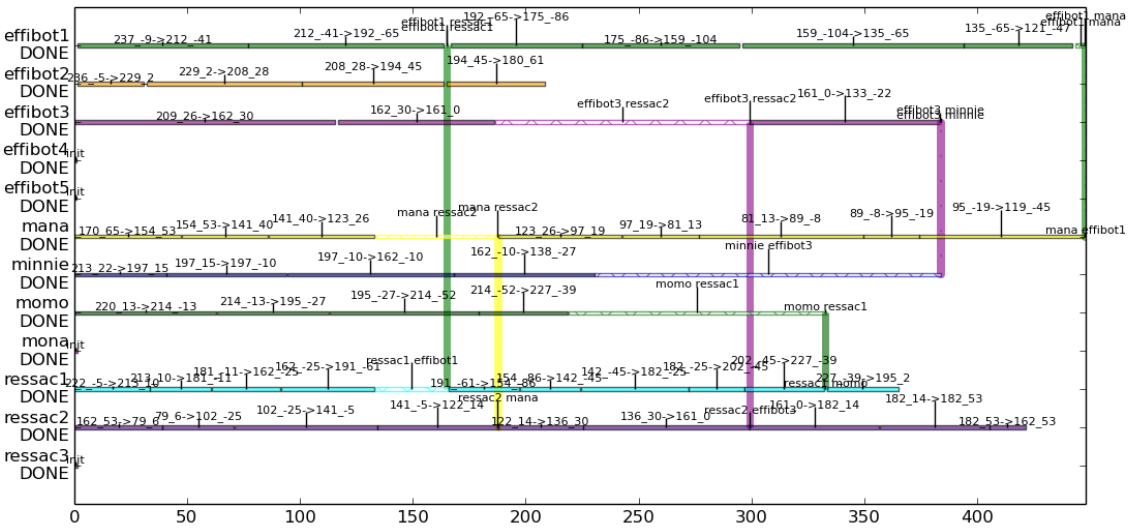


Figure E.15 – Timeline de la mission jouée VI.8.

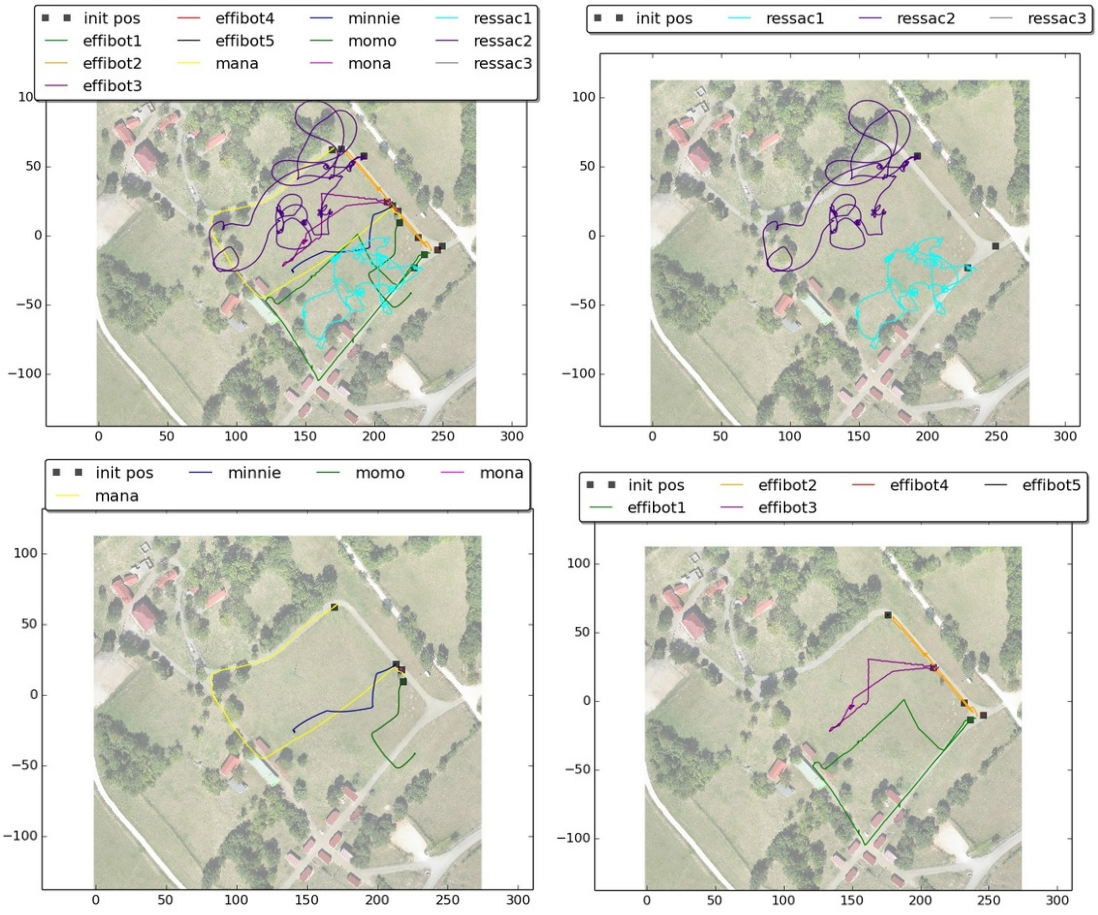


Figure E.16 – Itinéraires de la mission jouée VI.8 (véhicules réels et simulés).

Mission VI.9

Début de patrouille et deux cibles sont positionnées sur la zone de mission : une est suivie par Mana, l'autre par ReSSAC1. Aucun autre véhicule n'est sollicité pour le suivi de cible ; les actions des véhicules suiveurs sont reprises par deux véhicules de rechange. La patrouille se finit alors que les deux véhicules suivent toujours leur cible.

L'analyse de la timeline de la figure E.17 et des itinéraires de la figure E.18 montre les résultats suivants :

- Durée de la mission ∼9mn ;
- ReSSAC1 et Mana détectent et suivent chacun une cible ; les actions de ReSSAC1 sont reprises par le drone aérien simulé ReSSAC3 ; les actions de Mana sont reprises par Effibot5 ;
- ReSSAC2 est déclaré mort, ses actions ne sont pas reprises (par de véhicule de rechange pour lui) www ;
- Minnie et Momo finissent leur patrouille ;
- Effibot1, Effibot2 et Effibot3 finissent leur patrouille ;
- Un rendez-vous a lieu, ceux concernant les véhicules suivant les cibles étant annulés.

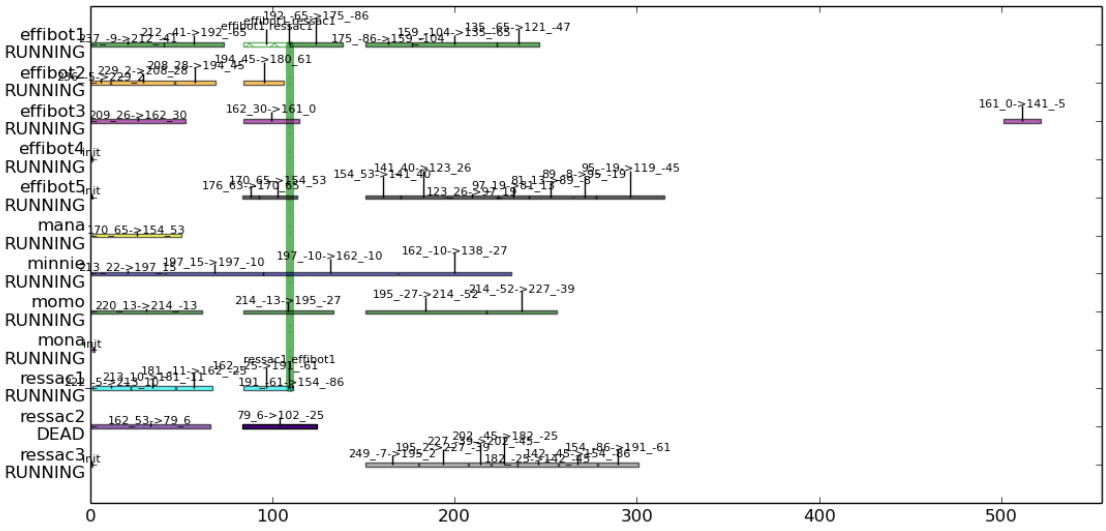


Figure E.17 – Timeline de la mission jouée VI.9.

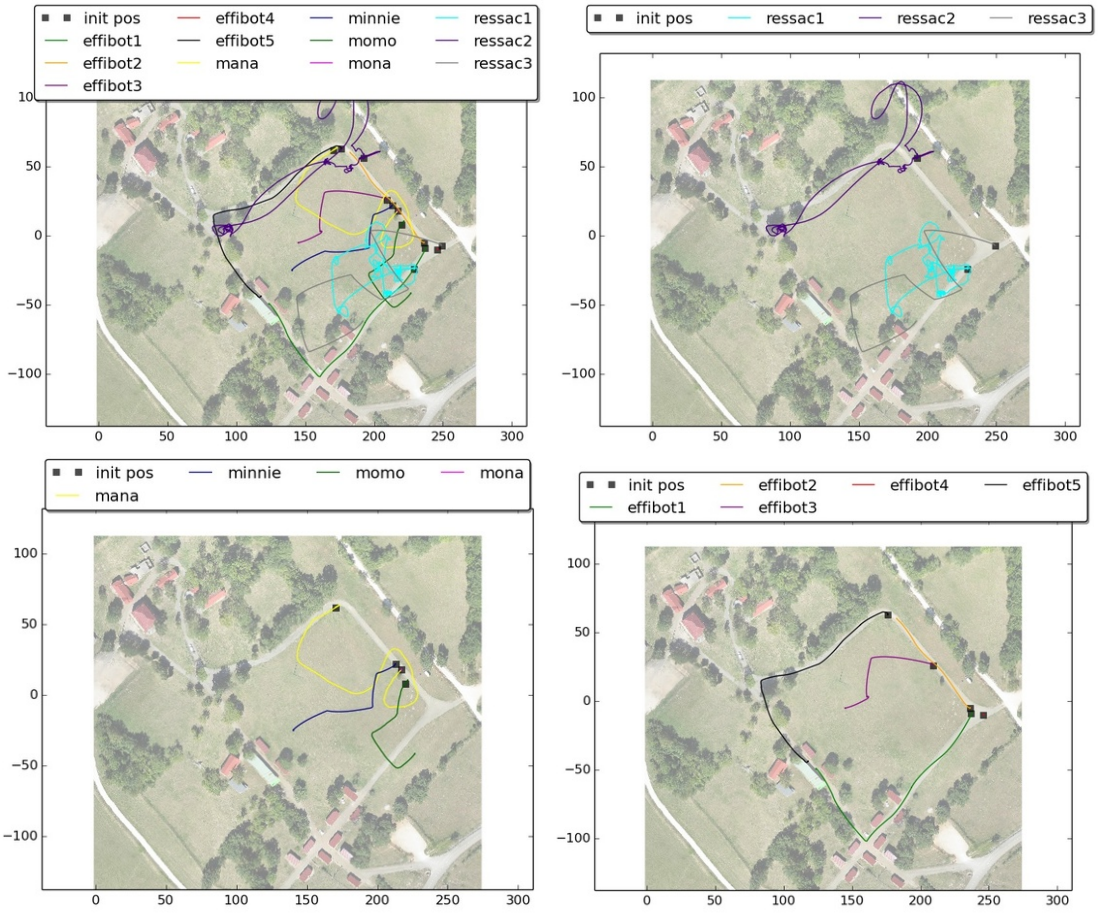


Figure E.18 – Itinéraires de la mission jouée VI.9 (véhicules réels et simulés).

Cinquième partie

Bibliographie

Bibliographie

Bacchus, Fahiem et Kabanza, Froduald (2000). « Using temporal logics to express search control knowledge for planning ». Dans : *Artificial Intelligence* 116.1-2, p. 123–191 (cité page 38).

Bajada, Josef, Fox, Maria et Long, Derek (2014). « Temporal Plan Quality Improvement and Repair using Local Search ». Dans : *Starting AI Researcher Symposium (STAIRS)* (cité page 44).

Barreiro, Javier et al. (2012). « EUROPA : A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 24 (cité page 26).

Bechon, Patrick et al. (2014). « HiPOP : Hierarchical Partial-Order Planning ». Dans : *Starting AI Researcher Symposium (STAIRS)* (cité page 3).

Bechon, Patrick et al. (2015). « Using hybrid planning for plan reparation ». Dans : *European Conference on Mobile Robots (ECMR)* (cité page 3).

Bercher, Pascal et al. (2014). « Plan, Repair, Execute, Explain - How Planning Helps to Assemble your Home Theater ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 386–394 (cité page 36).

Bidot, Julien, Schattenberg, Bernd et Biundo, Susanne (2008). « Plan Repair in Hybrid Planning ». Dans : *KI : Advances in Artificial Intelligence* 5243, p. 169–176 (cité page 47).

Blum, Avrim L et Furst, Merrick L (1997). « Fast planning through planning graph analysis ». Dans : *Artificial intelligence* (cité page 30).

Boella, Guido et Damiano, Rossana (2002). « A replanning algorithm for a reactive agent architecture ». Dans : *Artificial Intelligence : Methodology, Systems, and Applications* (cité page 45).

Bonet, Blai et Geffner, Héctor (2001). « Planning as heuristic search ». Dans : *Artificial Intelligence* (cité page 31).

Browne, Cameron B et al. (2012). « A Survey of Monte Carlo Tree Search Methods ». Dans : *IEEE Transactions on Computational Intelligence and AI in Games* 4.1, p. 1–43 (cité page 50).

Carlési, Nicolas et al. (2011). « Generic architecture for multi-AUV cooperation based on a multi-agent reactive organizational approach ». Dans : *Intelligent Robots and Systems (IROS)*, p. 5041–5047 (cité page 2).

Casanova, Guillaume, Pralet, Cédric et Lesire, Charles (2015). « Managing Dynamic Multi-Agent Simple Temporal Network ». Dans : *Autonomous Agents and Multi-Agent Systems (AAMAS)*. International Foundation for Autonomous Agents et Multiagent Systems, p. 1171–1179 (cité page 161).

Castillo, Luis et al. (2006). « Efficiently handling temporal knowledge in an HTN planner ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 63–72 (cité page 38).

Cesta, Amedeo et Oddi, Angelo (1996). « Gaining Efficiency and Flexibility in the Simple Temporal Problem ». Dans : *Third International Workshop on Temporal Representation and Reasoning*, p. 45–50 (cité page 158).

Chien, Steve et al. (2000). « Using Iterative Repair to Increase the Responsiveness of Planning and Scheduling ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 300–307 (cité pages 41, 51).

Cox, Michael, Muñoz-Avila, Héctor et Bergmann, Ralph (2005). « Case-based planning ». Dans : *The Knowledge Engineering Review* (cité page 41).

Currie, Ken et Tate, Austin (1991). « O-Plan : the open planning architecture ». Dans : *Artificial Intelligence* (cité page 37).

Cushing, William et Kambhampati, Subbarao (2005). « Replanning : A new perspective ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)* (cité page 42).

Dechter, Rina, Meiri, Itay et Pearl, Judea (1991). « Temporal constraint networks ». Dans : *Artificial intelligence* 49.1, p. 61–95 (cité page 32).

Do, Minh Binh et Kambhampati, Subbarao (2000). « Solving planning-graph by compiling it into CSP ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 82–91 (cité page 25).

Drabble, Brian, Dalton, Jeff et Tate, Austin (1997). « Repairing plans on-the-fly ». Dans : *Proceedings of the NASA Workshop on Planning and Scheduling for Space* (cité page 46).

Dvorak, Filip et al. (2014). « Planning and Acting with Temporal and Hierarchical Decomposition Models ». Dans : *International Conference on Tools with Artificial Intelligence (ICTAI)*, p. 115–121 (cité pages 26, 37, 58).

Echeverria, Gilberto et al. (2011). « Modular open robots simulation engine : Morse ». Dans : *International Conference on Robotics and Automation (ICRA)*. IEEE, p. 46–51 (cité page 170).

Edelkamp, Stefan et Hoffmann, Jörg (2004). « PDDL2. 2 : The language for the classical part of the 4th international planning competition ». Dans : *International Planning Competition (IPC)* (cité page 17).

Elkawkagy, Mohamed et Bercher, Pascal (2012). « Improving Hierarchical Planning Performance by the Use of Landmarks ». Dans : *AAAI*, p. 1763–1769 (cité page 193).

Erol, Kutluhan, Hendler, James et Nau, Dana S (1994). « HTN planning : Complexity and expressivity ». Dans : *AAAI*, p. 1123–1128 (cité page 20).

Erol, Kutluhan, Nau, Dana S et Subrahmanian, V.S (1995). « Complexity, decidability and undecidability results for domain-independent planning ». Dans : *Artificial Intelligence* 76.1-2, p. 75–88 (cité page 27).

Estlin, Tara A, Chien, Steve et Wang, Xuemei (1997). « An Argument for a Hybrid HTN/Operator-Based Approach to Planning ». Dans : *European Conference on Planning*, p. 182–194 (cité page 38).

Fernández-Olivares, Juan et al. (2006). « Bringing Users and Planning Technology Together. Experiences in SIADEX ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 11–20 (cité page 2).

Fikes, Richard E et Nilsson, Nils J (1972). « STRIPS : A new approach to the application of theorem proving to problem solving ». Dans : *Artificial intelligence* 2.October, p. 189–208 (cité page 13).

Fox, Maria, Gerevini, Alfonso et al. (2006). « Plan stability : replanning versus plan repair ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 212–221 (cité page 41).

Fox, Maria et Long, Derek (2003). « PDDL2.1 : An extension to PDDL for expressing temporal planning domains ». Dans : *Journal of Artificial Intelligence Research (JAIR)* 20, p. 61–124. arXiv : 1106.4561 (cité pages 17, 19, 199).

Gateau, Thibault, Lesire, Charles et Barbier, Magali (2013). « HiDDeN : Cooperative Plan Execution and Repair for Heterogeneous Robots in Dynamic Environments ». Dans : *International Conference on Intelligent Robots and Systems (IROS)*. Tokyo, Japan (cité page 1).

Gerevini, Alfonso et Long, Derek (2006). « Preferences and Soft Constraints in {PDDL3} ». Dans : *Proceedings of the ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning*, p. 46–54 (cité pages 17, 199).

Gerevini, Alfonso et Serina, Ivan (2000). « Fast Plan Adaptation through Planning Graphs : Local and Systematic Search Techniques ». Dans : *Artificial Intelligence Planning Systems (AIPS)*, p. 112–121 (cité pages 42, 43).

Ghallab, Malik, Nau, Dana S et Traverso, Paolo (2004). *Automated planning : theory & practice* (cité page 28).

Hammond, Kristian J (1990). « Explaining and repairing plans that fail ». Dans : *Artificial intelligence* (cité page 46).

Hansen, Eric A et Zilberstein, Shlomo (2001). « LAO* : a heuristic search algorithm that finds solutions with loops ». Dans : *Artificial Intelligence* 129.1-2, p. 35–62 (cité page 50).

Hart, Peter E, Nilsson, Nils J et Raphael, Bertram (1968). « A Formal Basis for the Heuristic Determination of Minimum Cost Paths ». Dans : *IEEE Transactions on Systems Science and Cybernetics* 4.2, p. 100–107 (cité page 73).

Hashmi, Muhammad Adnan et Seghrouchni, Amal El Fallah (2010). « Merging of Temporal Plans Supported by Plan Repairing ». Dans : *International Conference on Tools with Artificial Intelligence (ICTAI)* (cité pages 36, 57).

Haslum, Patrik et Geffner, Héctor (2000). « Admissible Heuristics for Optimal Planning. » Dans : *Artificial Intelligence Planning Systems (AIPS)*. Citeseer, p. 140–149 (cité page 32).

Helmert, Malte (2009). « Concise finite-domain representations for PDDL planning tasks ». Dans : *Artificial Intelligence* 173.5, p. 503–535 (cité pages 59, 193).

Hoffmann, Jörg et Nebel, Bernhard (2011). « The FF planning system : Fast plan generation through heuristic search ». Dans : *Journal of Artificial Intelligence Research (JAIR)* (cité page 31).

Hoffmann, Jörg, Porteous, Julie et Sebastia, Laura (2004). « Ordered landmarks in planning ». Dans : *Journal of Artificial Intelligence Research (JAIR)* (cité page 29).

Howey, Richard et Long, Derek (2003). « VAL's Progress : The Automatic Validation Tool for PDDL2.1 Used in the International Planning Competition ». Dans : *Proceedings of the ICAPS Workshop on The Competition : Impact, Organization, Evaluation, Benchmarks*, p. 28–37 (cité page 58).

Jónsson, Ari et al. (2000). « Planning in interplanetary space : Theory and practice ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 177–86 (cité page 51).

Kambhampati, Subbarao, Knoblock, Craig et Yang, Qiang (1995). « Planning as refinement search : a unified framework for evaluating design tradeoffs in partial-order planning ». Dans : *Artificial Intelligence* 76.1-2, p. 167–238 (cité page 28).

Kambhampati, Subbarao, Mali, Amol et Srivastava, Biplav (1998). « Hybrid planning for partially hierarchical domains ». Dans : *AAAI* c, p. 882–888 (cité page 37).

Koenig, Sven, Furcy, David et Bauer, Colin (2002). « Heuristic Search-Based Replanning. » Dans : *Artificial Intelligence Planning Systems (AIPS)* (cité page 42).

Kovacs, Daniel L (2012). « A Multi-Agent Extension of PDDL3.1 ». Dans : *WS-IPC 2012* (cité pages 17, 142).

Krogt, Roman Van Der et Weerdt, Mathijs De (2005). « Plan Repair as an Extension of Planning ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)*, p. 161–170 (cité pages 41, 42, 45, 129).

Kushmerick, Nicholas, Hanks, Steve et Weld, Daniel S (1995). « An algorithm for probabilistic planning ». Dans : *Artificial Intelligence* 76.1-2, p. 239–286 (cité page 50).

Kvarnström, Jonas et Doherty, Patrick (2000). « TALplanner : A temporal logic based forward chaining planner ». Dans : *Annals of Mathematics and Artificial Intelligence* 30.1, p. 119–169 (cité page 38).

Lee-Urban, Stephen (2012). « Hierarchical Planning Knowledge for Refining Partial-Order Plans ». Thèse de doct. Lehigh University (cité page 41).

Lemai-Chenevier, Solange (2004). « IXTET-EXEC : planning, plan repair and execution control with time and resource management ». Thèse de doct. CNRS-LAAS (cité pages 26, 50).

Marthi, Bhaskara, Russell, Stuart J et Wolfe, Jason (2007). « Angelic semantics for high-level actions ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)* (cité page 37).

Mcdermott, Drew et al. (1998). *PDDL-the planning domain definition language*. Rapp. tech. (cité page 15).

McGann, Conor et al. (2008). « A deliberative architecture for AUV control ». Dans : *International Conference on Robotics and Automation (ICRA)*. IEEE, p. 1049–1054 (cité page 51).

Morris, Paul, Muscettola, Nicola et Vidal, Thierry (2001). « Dynamic control of plans with temporal uncertainty ». Dans : *International Joint Conference on Artificial Intelligence (IJCAI)*, p. 494–499 (cité page 196).

Muñoz-Avila, Héctor et Weberskirch, Frank (1996). « Planning for manufacturing workpieces by storing, indexing and replaying planning decisions ». Dans : *Artificial Intelligence Planning Systems (AIPS)* (cité pages 42, 47).

Nakhost, Hootan et Müller, Martin (2010). « Action Elimination and Plan Neighborhood Graph Search : Two Algorithms for Plan Improvement. » Dans : *International Conference on Automated Planning & Scheduling (ICAPS)* (cité page 43).

Nau, Dana S, Au, Tsz Chiu et al. (2003). « SHOP2 : An HTN planning system ». Dans : *Journal of Artificial Intelligence Research (JAIR)* 20.1, p. 379–404. arXiv : 1106.4869 (cité page 37).

Nau, Dana S, Cao, Yue et al. (1999). « SHOP : Simple hierarchical ordered planner ». Dans : *International Joint Conference on Artificial Intelligence (IJCAI)* (cité page 36).

Nebel, Bernhard (2000). « On the compilability and expressive power of propositional planning formalisms ». Dans : *Journal of Artificial Intelligence Research (JAIR)* (cité page 26).

Nguyen, XuanLong et Kambhampati, Subbarao (2001). « Reviving partial order planning ». Dans : *International Joint Conference on Artificial Intelligence (IJCAI)*, p. 459–464 (cité page 35).

Nilsson, Nils J (1984). *Shakey the Robot. Technical Note 323. SRI AI Center.* Rapp. tech. (cité pages 1, 13).

Penberthy, J Scott et Weld, Daniel S (1992). « UCPOP : A sound, complete, partial order planner for ADL ». Dans : *KR* (cité pages 29, 35).

Peot, Mark A et Smith, David E (1992). « Conditional nonlinear planning ». Dans : *Artificial Intelligence Planning Systems (AIPS)* (cité page 48).

Pineau, Joelle, Gordon, Geoff et Thrun, Sebastian (2003). « Point-based value iteration : An anytime algorithm for POMDPs ». Dans : *International Joint Conference on Artificial Intelligence (IJCAI)*, p. 1025–1030 (cité page 50).

Planken, Léon, Weerdt, Mathijs De et Yorke-Smith, Neil (2010). « Incrementally solving STNs by enforcing partial path consistency ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)* (cité page 158).

Puterman, Martin L (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. 1st. New York, NY, USA : John Wiley et Sons, Inc. (cité page 49).

Richter, Silvia, Helmert, Malte et Westphal, Matthias (2008). « Landmarks Revisited. » Dans : *AAAI* (cité page 29).

Richter, Silvia et Westphal, Matthias (2010). « The LAMA planner : Guiding cost-based anytime planning with landmarks ». Dans : *Journal of Artificial Intelligence Research (JAIR)* (cité page 29).

Röger, Gabriele, Eyerich, Patrick et Mattmüller, Robert (2008). « TFD : A numeric temporal extension to Fast Downward ». Dans : *International Planning Competition (IPC)* (cité page 89).

Schattenberg, Bernd (2009). « Hybrid Planning And Scheduling ». Thèse de doct. Ulm University, Institute of Artificial Intelligence (cité pages 37, 58).

Silver, David et Veness, Joel (2010). « Monte-Carlo Planning in large POMDPs ». Dans : *Advances in Neural Information Processing Systems*. Sous la dir. de J D Lafferty et al. Curran Associates, Inc., p. 2164–2172 (cité page 50).

Smallwood, Richard D et Sondik, Edward J (1973). *The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon*. T. 21. INFORMS, p. 1071–1088 (cité page 50).

Smith, David E (2004). « Choosing Objectives in Over-Subscription Planning. » Dans : *International Conference on Automated Planning & Scheduling (ICAPS)* (cité page 27).

Smith, David E, Frank, Jeremy et Cushing, William (2008). « The ANML language ». Dans : *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling* (cité pages 26, 199).

Smith, Trey et Simmons, Reid (2004). « Heuristic search value iteration for POMDPs ». Dans : *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. UAI '04. Arlington, Virginia, United States : AUAI Press, p. 520–527 (cité page 50).

Stulp, Freek et Beetz, Michael (2005). « Optimized execution of action chains using learned performance models of abstract actions ». Dans : *International Joint Conference on Artificial Intelligence (IJCAI)*, p. 1272–1277 (cité page 41).

Vernhes, Simon (2014). « Décomposition des problèmes de planification de tâches basée sur les landmarks ». Thèse de doct. Université de Toulouse (cité page 29).

Vidal, Vincent (2011). « YAHSP2 : Keep it simple, stupid ». Dans : *International Planning Competition (IPC)*, p. 83–90 (cité pages 31, 89).

Vossen, Thomas et al. (2005). « On the Use of Integer Programming Models in AI Planning ». Dans : *Mathematica*, p. 1–4 (cité page 25).

Wang, Xuemei et Chien, Steve (1997). *Replanning using hierarchical task network and operator-based planning*. Rapp. tech., p. 427–439 (cité page 46).

Warfield, Ian et al. (2007). « Adaptation of Hierarchical Task Network Plans ». Dans : *FLAIRS Conference*, p. 429–434 (cité pages 42, 47).

Younes, Hakan et Littman, Michael L (2004). « PPDDL1. 0 : An extension to PDDL for expressing planning domains with probabilistic effects ». Dans : *International Conference on Automated Planning & Scheduling (ICAPS)* (cité page 50).

Younes, Hakan et Simmons, Reid (2002). « On the role of ground actions in refinement planning ». Dans : *Artificial Intelligence Planning Systems (AIPS)*, p. 54–61 (cité page 29).

Younes, Hakan et Simmons, Reid (2003). « VHPOP : Versatile heuristic partial order planner ». Dans : *Journal of Artificial Intelligence Research (JAIR)* 20, p. 405–430 (cité pages 35, 36, 73, 89).

Young, R Michael, Pollack, Martha E et Moore, Johanna D (1994). « Decomposition and causality in partial-order planning ». Dans : *Artificial Intelligence Planning Systems (AIPS)* (cité page 37).

Titre Planification multirobot pour des missions de surveillance avec contraintes de communication

Résumé L'objectif de ce travail est de permettre à une équipe de robots autonomes hétérogènes d'effectuer une mission complexe dans un environnement réel et sous contrainte de communication. Cette thèse a donc consisté à créer et à valider une architecture distribuée à bord des robots et intégrant planification, supervision de l'exécution du plan et réparation de ce plan suite à l'occurrence d'aléas. Ce manuscrit présente la conception d'un algorithme de planification hybride, dénommé HiPOP, utilisé pour calculer un plan initial, avant le début de la mission, et pour réparer le plan en cours de mission quand un événement perturbateur survient. Il présente aussi la conception d'un algorithme de supervision, dénommé METAL, utilisé pour suivre l'exécution du plan sur chaque robot et, le cas échéant, faisant appel à HiPOP pour réparer le plan. Ces deux algorithmes ont été implémentés et ont permis de réaliser des missions de surveillance allant jusqu'à impliquer 12 robots, à la fois en simulation et avec de vrais robots.

Mots-clés Planification, supervision, réparation de plan, coopération multirobot, robotique autonome

Title Multirobot planning for surveillance missions with communication constraints

Abstract The goal of this work is to enable a team of heterogeneous autonomous robots to perform a complex mission in a real environment with communication constraints. This approach was therefore to create and validate a distributed embedded architecture able to plan, to monitor the execution of a plan and to repair a plan when an unexpected event occurs. This document shows the conception of an hybrid planning algorithm, named HiPOP, used to compute initial plans before the beginning of the mission and to repair the plan during the mission when something unexpected happens. It also shows the conception of a monitoring algorithm, named METAL, used to monitor the execution of the plan on each robot and, when needed, which calls HiPOP to repair the plan. Both algorithms were implemented and used to carry out surveillance missions up to 12 robots, both in simulation and in a real life scenario.

Keywords Planning, plan execution, plan repair, multirobot collaboration, autonomous robots