Modèles et algorithmes pour systèmes multi-robots hétérogènes : application à la patrouille et au suivi de cible

Cyril Robin

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par

l'Institut National des Sciences Appliquées (INSA) de Toulouse

---

Présentée et soutenue le 4 juin 2015 *par*

Cyril Robin

Modèles et Algorithmes pour systèmes multirobots hétérogènes

Application à la patrouille et au suivi de cible

École doctorale et discipline ou spécialité :
EDSYS : Robotique 4200046

Unité de recherche : CNRS LAAS

Directeur de thèse
M. Simon Lacroix

Jury

Rapporteurs	Mme Amal El Fallah Segrouchni	Professeur
	M. Olivier Simonin	Professeur
Examinateurs	Mme Véronique Serfaty	Docteur
	M Noury Bouraquadi	Professeur
	M. Rachid Alami	DR CNRS
	M. Simon Lacroix	DR CNRS

Thèse

Préparée au LAAS-CNRS
Sous la direction de Simon Lacroix

Modèles et Algorithmes pour systèmes multirobots hétérogènes

Application à la patrouille et au suivi de cible

Cyril Robin

Présentée et soutenue le 6 juin 2015
en vue de l'obtention du

Doctorat de l'Université de Toulouse

délivré par l'INSA Toulouse

Ecole Doctorale Sytèmes

Robotique et Informatique

Rapporteurs	Amal El Fallah Seghrouchni (LIP6, UPMC) Olivier Simonin (CITI-INRIA)
Examinateurs	Noury Bouraqadi (Mines Douai) Véronique Serfaty (DGA) Rachid Alami (LAAS-CNRS) Simon Lacroix (LAAS-CNRS)

À Sophie, Didier, Ginette, Lydie, Roger, Armande et Jean.
À Christophe.

Ce qui vient au monde pour ne rien troubler
ne mérite ni égards ni patience.

— René Char

RÉSUMÉ

La détection et le suivi de cibles sont des missions fréquentes pour la robotique mobile, que le contexte soit civil, industriel ou militaire. Ces applications constituent un domaine de choix pour la planification multirobot, et sont abordées par de multiples communautés selon différents points de vue.

Nous proposons dans un premier temps une taxonomie commune qui permet de regrouper et de comparer les différentes approches de ces problèmes, afin de mieux les analyser et de mettre en évidence leurs lacunes respectives. En particulier, on note la faible représentativité des modèles exploités, peu expressifs : la plupart des algorithmes évoluent dans un monde en deux dimensions où les observations et le déplacement sont conditionnés par les mêmes obstacles. Ces modèles éloignés de la réalité nous semblent trop restrictifs pour pleinement exploiter la synergie des équipes multirobot hétérogènes : nous proposons une organisation des différents modèles nécessaires, en explicitant une séparation claire entre modèles et algorithmes de planification. Cette organisation est concrétisée par une librairie qui structure les modèles disponibles et définit les requêtes nécessaires aux algorithmes de planification.

Dans un second temps, nous proposons un ensemble d'algorithmes utilisant les modèles définis précédemment pour planifier des missions de patrouille de zones et de poursuite de cibles. Ces algorithmes s'appuient sur un formalisme mathématique rigoureux afin d'étudier l'impact des modèles sur les performances. Nous analysons notamment l'impact sur la complexité – c'est-à-dire en quoi des modèles plus élaborés impactent la complexité de résolution – et sur la qualité des solutions résultantes, indépendamment des modèles, selon des métriques usuelles.

D'une manière plus générale, les modèles sont un lien essentiel entre l'Intelligence Artificielle et la Robotique : leur enrichissement et leur étude approfondie permettent d'exhiber des comportements plus efficaces pour la réussite des missions allouées aux robots. Cette thèse contribue à démontrer l'importance des modèles pour la planification et la conduite de mission multirobots.

Mots-clefs : Systèmes multirobots ; Détection de cibles ; Suivi de cibles ; Missions de patrouille ; Modèles pour la planification.

Models and Algorithms for heterogeneous multi-robot systems

Applied to patrolling and target tracking

Cyril Robin

ABSTRACT

Detecting, localizing or following targets is at the core of numerous robotic applications in industrial, civilian and military application contexts. Much work has been devoted in various research communities to planning for such problems, each community with a different standpoint.

Our thesis first provides a unifying taxonomy to go beyond the frontiers of specific communities and specific problems, and to enlarge the scope of prior surveys. We review various work related to each class of problems identified in the taxonomy, highlighting the different approaches, models and results. This analysis specifically points out the lack of representativity of the exploited models, which are in vast majority only 2D single-layer models where motion and sensing are mixed up. We consider those unrealistic models as too restrictive to handle the full synergistic potential of an heterogeneous team of cooperative robots. In response to this statement, we suggest a new organisation of the necessary models, stating clearly the links and separation between models and planning algorithms. This has lead to the development of a C++ library that structures the available models and defines the requests required by the planning process.

We then exploit this library through a set of algorithms tackling area patrolling and target tracking. These algorithms are supported by a sound formalism and we study the impact of the models on the observed performances, with an emphasis on the complexity and the quality of the resulting solutions.

As a more general consideration, models are an essential link between Artificial Intelligence and applied Robotics : improving their expressiveness and studying them rigorously are the keys leading toward better robot behaviours and successful robotic missions. This thesis help to show how important the models are for planning and other decision processes for multi-robot missions.

Keywords : Multi-robot Systems ; Target Detection ; Patrolling ; Target Tracking ; Models for planning.

PUBLICATIONS DE L'AUTEUR

Certains travaux, idées et figures présentés ci-après ont déjà fait l'objet de publications :

[1] Cyril Robin and Simon Lacroix, Failure anticipation on pursuit-evasion. *Robotics : Science and Systems conference 2012 (RSS VIII) - MIT Press*, 2012.

[2] Cyril Robin and Simon Lacroix, Anticipating failures in realistic pursuit-evasion. *14ème Congrès des doctorants EDSYS*, 2013.

[3] Cyril Robin and Simon Lacroix, Taxonomy on Multi-robot Target Detection and Tracking. *21st European Conference on Artificial Intelligence (ECAI'14), Workshop on Multi-Agent Coordination in Robotic Exploration (MACOREX)*, 2014.

[4] Cyril Robin and Simon Lacroix, Multi-robot Target Detection and Tracking : Taxonomy and Survey. *Autonomous robots*, 2015.

Eine feine Seele bedrückt es,
sich jemandem zum Dank verpflichtet zu wissen,
eine grobe, jemandem zu Dank verpflichtet zu sein.

— Friedrich Nietzsche

REMERCIEMENTS

Un œil averti peut percevoir dans une thèse, au delà du travail présenté, certains reflets de la personnalité de son auteur. À ce titre, il serait difficile de remercier toutes les personnes ayant contribué, par leur contact, leurs remarques, leurs conseils, ou leur exemple, à façonner l'auteur de ces lignes. Aussi, à défaut de faire une liste exhaustive, ces remerciements seront concentrés sur ceux qui, plus spécifiquement, m'ont côtoyé pendant ces presque quatre années passées au LAAS.

Ainsi, dans ce contexte professionnel, c'est vers toi, *Simon*, que se tournent mes premiers remerciements, pour m'avoir accueilli et fait confiance, pour la qualité de nos rapports humains, pour nos échanges scientifiques, et pour m'avoir donné le plus grand exercice de version de toute ma vie ! J'aimerais également remercier l'équipe RIS, et particulièrement ses doctorants et post-doctorants, pour l'ambiance au laboratoire et en dehors ; vous allez me manquer – en fait, vous me manquez déjà. Merci aussi à l'équipe de *field robotics*, pour les moments partagés autour d'un café ou en vadrouille sur le terrain : *Baptiste*, *Ellon*, *Artur*, *Pierrick*, *Cyril* et *Arnaud* (nos discussions me manquent !). I also would like to thank *Thierry*, *Marcos* and *Robert* for their warmful welcome down under. And thank you *Montse*, for your complicity, and for being a genuine smiling sunshine that cheered me up !

Sur un plan plus personnel et privé, j'aimerais vous remercier, *Papa*, *Maman*, pour votre soutien, et pour n'avoir pas relâcher votre travail depuis maintenant 28 ans. *Sophie*, je ne sais pas si tu t'en aperçois, mais tu fais bien plus que partager mon quotidien : tu me rappelles que les belles choses sont parfois tout près de nous, et qu'il suffit de quitter des yeux quelques instants l'horizon pour les voir. Pour tout cela comme pour le reste, je te remercie. Merci aussi à toi, *Christophe* ; merci tout simplement d'être là et d'être toi.

Merci aussi à vous, mes amis, qui êtes souvent loin, ou plutôt dont je suis souvent trop loin, mais que j'emmène partout avec moi, au fond de moi : *Florent* (pour tout ce que nous avons partagé, la flûte et les après-midi crèpes et jeux ! et pour être encore là après tant d'années), *Will* (pour ta fidélité en amitié et ton humour qui me fait toujours rire), *Rémi* (pour me rappeler qu'il existe d'autres façons de voir et de faire, et pour tes maudites

contrepèteries !), *Élise* (pour ton énergie à rassembler les gens, ton caractère et ta cuisine !), *Chloé* (pour les 1er de l'an à Toulouse, les déjeuners à Paris, tous ces moments précieux, pour ton écoute et ta douceur), *Angèle* (pour être la famille que l'on s'est choisi), *Céline* (pour ta créativité et ton énergie), *Hélène* (pour ta poésie et ton regard sur le monde), *Max* (pour les parties de cartes, ta bonne humeur, ton enthousiasme, et pour savoir gentiment me ramener sur terre parfois) *Thomas* (pour m'avoir fait découvrir WoW en prépa (!), pour ton côté râleur grognon affectueux, qui cache un caractère engagé et passionné), *Alice* (pour nos échanges sans paroles, pour ta sensibilité, ton attention aux choses et aux personnes qui t'entourent, et pour ton courage), *Mathieu* (pour m'avoir accompagné dans le désert, pour m'écouter même quand je parle trop, et pour ta franchise), *Marion* (pour avoir accepté d'être ma Marraine, et pour l'être encore, des années après ! pour ton énergie, ta capacité à rassembler les gens, et à faire avancer positivement les choses), *Manon* (pour ton rire et la joie qu'il fait naître), *Antoine* (pour avoir partagé avec complicité les mêmes bancs d'école pendant quatre année), *Bertrand* (pour les souvenirs du hand, ta bonhomie naturelle et débordante, et ton enthousiasme qui vient soutenir ton sens des responsabilités), *Irwin* (en souvenir des soirées « natation », pour m'avoir fait aimer la Normandie (le Nord !), et pour un amour partagé des bonnes choses et des bons moments), *Béa* (pour ton énergie et ton amour des confitures), *Olivier* (pour ton écoute, ta sagacité et ta volonté), *Carine* (pour ton empathie, ta complicité et ton énergie), *Benjamin* (pour ton humour et ta discipline), *Lucie* (pour ta sensibilité, ton ingéniosité et tes improvisations au piano), *Cécile* (pour être venu me chercher et m'avoir supporté deux ans), *Lucie* (pour ta présence, ta discrétion, ta sensibilité et ton enthousiasme, ta fragilité forte), *Gaël* (pour ton sourire, ta sensibilité artistique et ta force intérieure), *Virginie* (pour les balades en roller, pour ton énergie positive, et ta capacité à voir les choses simplement), Merci à vous tous pour tous les moments passés ensemble, et pour ceux à venir – nombreux j'espère. Merci pour ce que vous m'inspirez.

Last but not least, merci enfin à vous autres, perdus de vue ou non, qui m'avez accompagné, changé, fait voir le monde, aimé et que j'ai aimés, pour les morceaux de vie partagés. Pour ces petits bouts de moi que je vous dois.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

LISTE DES EXTRAITS DE CODE SOURCE

Nous faisons dans ce document un usage fréquent de notes annexes, situées dans les marges et auxquelles le lecteur est renvoyé à chaque astérisque (*). Ces notes sont un complément au texte principal (citations, éclaircissements, vocabulaire, remarques accessoires, *etc.*).

INTRODUCTION

Beau mot que celui de chercheur, et si préférable à celui de savant !
Il exprime la saine attitude de l'esprit devant la vérité :
le manque plus que l'avoir, le désir plus que la possession,
l'appétit plus que la satiété.

— Jean Rostand

La détection et le suivi de cibles sont des missions fréquemment considérées en robotique mobile, que le contexte soit civil, industriel ou militaire. Ce sont souvent des missions répétitives et longues (surveillance, patrouille, observation, *etc.*) présentant parfois un contexte dangereux pour l'Homme. Leur robotisation est une réponse future à tous ces problèmes. Ces applications tirent par ailleurs un large bénéfice du développement de la planification multirobot, permettant de couvrir de plus larges zones et d'exploiter des complémentarités entre robots. La planification multirobot considère principalement deux grands axes : des équipes de grande taille constituées de robots identiques et peu onéreux, et des équipes hétérogènes de taille réduite constituées de robots différents mais généralement plus performants individuellement. Les travaux présentés ci-après se placent dans ce dernier cadre et se concentrent sur des missions de patrouille et de suivi de cible.

Les travaux présentés par la suite ont pour cadre l'équipe de robotique mobile du LAAS-CNRS sous la direction de Simon Lacroix. Le principal cadre applicatif le PEA ACTION*, mettant en œuvre des équipes robotiques aéroterrestres de taille réduite pour des scénarios de surveillance. Ce cadre a motivé les applications principales de nos travaux : la patrouille de zones sécurisées et le suivi de cibles par des petites équipes de robots hétérogènes. Les robots considérés sont des véhicules aériens autonomes (AAVs) et des véhicules terrestres autonomes (AGVs) – figure 1.

Le PEA ACTION est un programme financé par la DGA, et dont l'Onera et le LAAS-CNRS sont co-titulaires.

(a) AGVs

(b) AAV

FIGURE 1 – Plateformes mobiles utilisées au sein de l'équipe de robotique mobile du LAAS (a) et de l'Onera (b).

Les contributions de cette thèse sont :

— La proposition d'une taxonomie des problèmes de détection et suivi de cibles, transverse aux différentes communautés. Cette taxonomie est orientée « problème » et nous sert de grille d'analyse pour un état de l'art de ce large sujet. Nous proposons une analyse synthétique de cet état de l'art.
— Une organisation et une formalisation des modèles de l'environnement et des robots nécessaires à la planification de leurs actions.
— Une approche de la planification de missions de patrouille en contexte antagoniste par des techniques d'optimisation en nombres entiers – ou *Integer Programming* (IP).
— Une approche du suivi de cible mobile par un ensemble de robots, basée sur un principe d'économie de ressources.

La suite de cette thèse est organisée ainsi : une première partie rassemble nos considérations générales sur la planification multirobot, présentant d'abord notre taxonomie – chapitre 1 – puis l'état de l'art associé dans les domaines de la détection – chapitre 2 – et du pistage de cible – chapitre 3. Cet état de l'art est suivi de notre analyse synthétique des travaux le constituant – chapitre 4. Un bilan conclut cette partie I en mettant en avant trois grandes directions dans lesquelles nous encourageons la communauté scientifique à poursuivre voire accentuer ses efforts de recherche.

Ces directions guident nos travaux algorithmiques, présentés dans la partie II. Nous présentons d'abord une organisation et une formalisation générale des modèles exploités par les processus de planification – chapitre 5. La suite de cette partie algorithmique exploite ces modèles à travers deux grandes applications : la patrouille de zones sécurisées et le suivi de cible. Le chapitre 6 expose nos considérations théoriques et notre approche de la patrouille de zones en contexte antagoniste ; les résultats expérimentaux associés sont présentés aux chapitres 7 et 8, ce dernier étudiant spécifiquement l'apport d'une approche décentralisée. Enfin, le chapitre 9 propose une nouvelle approche du suivi de cible sous contraintes de ressources. Une discussion de nos travaux et des perspectives à court et long termes conclut ce document.

PUBLICATIONS ASSOCIÉES

Les travaux associés à la partie I font l'objet de deux publications [151, 150], la plus récente étant en cours de révision après un premier retour positif. Le chapitre 5 fait écho à une librairie de modèles développée en équipe avec Pierrick Koch. Les chapitres 6, 7 et 8 n'ont pas encore fait l'objet de publications, les résultats associés étant très récents. Le chapitre 9 a donné lieu a deux publications [148, 149].

Première partie

CONSIDÉRATIONS GÉNÉRALES SUR LA PLANIFICATION MULTIROBOT

Dans cette première partie, nous posons les bases de notre réflexion sur la planification pour systèmes multirobots. Nous nous intéressons particulièrement à la gestion de cibles, rassemblant une large variété de problèmes. En s'appuyant sur une nouvelle taxonomie des problèmes de détection et de pistage de cibles présentée au chapitre 1, nous détaillons un état de l'art transverse aux différentes communautés – chapitres 2 et 3. Le chapitre 4 synthétise les approches et modèles actuels, dont nous analysons la portée et les limites. Le bilan de cette analyse conclut cette première partie et guidera nos travaux présentés en partie II.

1 ÉTAT DE L'ART : TAXONOMIE

Quot homines, tot sententiae.
(Autant d'hommes, autant d'opinions.)

— Térence

Ce chapitre présente une taxonomie des problèmes de détection et de suivi de cibles*. Nous avons établi cette taxonomie comme la base d'un état de l'art du domaine, dont le but est d'aller au-delà des frontières de chaque communauté ou de certains problèmes spécifiques : l'état de l'art développé dans les chapitres 2 et 3 s'articule autour de cette taxonomie « orientée problèmes », qui constitue le lieu de rassemblement et de comparaison d'un large ensemble de travaux produits par différentes communautés ces dernières décennies, avec une emphase sur les travaux multirobot produits lors de la dernière décennie.

Ces travaux, développés tout au long de la première partie, ont fait l'objet de deux publications [151, 150]

1.1 MOTIVATIONS

La détection, la localisation et le suivi de cibles sont au cœur de nombreuses applications en robotique, tant dans des contextes antagonistes que coopératifs. Ces problèmes ont fait l'objet de nombreux travaux dans différentes communautés de recherche, principalement sous les termes de *problèmes de « poursuite-évasion »*. Sous ce terme évocateur se cachent en fait une grande variété de scénarios : mono- ou multirobot, considérant une ou plusieurs cibles, dans le but de les détecter, de les capturer ou simplement de les suivre.

Par ailleurs, on trouve dans la littérature de nombreux problèmes similaires sous des noms différents, comme la surveillance, la fouille (*search*) ou le pistage (*tracking*), chacun utilisant un champ lexical spécifique. Ceci s'explique en partie par les différentes applications considérées (industrielles, civiles ou militaires), et en partie par les différentes communautés abordant ces problèmes depuis différents points de vue (planification symbolique ou géométrique, commande, capteurs, théorie des jeux, allocation de tâches*, *etc.*).

L'allocation de tâches consiste à répartir un ensemble de tâches entre plusieurs agents en cherchant à optimiser certains critères (ex : le temps global pour effectuer les tâches).

Face à l'étendue des problèmes de robotique relatifs à la notion de cible, on dénombre un grand nombre de contributions et un large éventail d'approches. Plusieurs études proposent une vue d'ensemble sur quelques problèmes spécifiques, en rassemblant les travaux qui leur sont relatifs [169, 10, 41, 102, 137, 119, 163]. Elles forment de bons points d'entrée pour le lecteur désireux d'approfondir ses connaissances sur un problème précis ou sur le point de vue d'une communauté. Dans ce chapitre, notre objectif

est d'aller au-delà des frontières entre ces communautés et au-delà des spécificités des problèmes rencontrés, afin d'élargir le champ des études déjà existantes. La détection et le suivi de cibles, qui forment les deux grandes catégories de scénarios relatifs aux cibles, ont *a priori* peu de choses en commun et sont abordés séparément, comme deux phases distinctes. Pourtant, en pratique, ces scénarios sont généralement entremêlés et sont joués par les mêmes équipes de robots : c'est pourquoi nous considérons qu'il est pertinent de les analyser et de les traiter de pair.

Ce chapitre présente une taxonomie des problèmes relatifs aux cibles. Elle a été élaborée dans le but de rassembler les différentes communautés étudiant des problèmes similaires, et propose un vocabulaire unifiant. Elle montre aussi comment les différents problèmes s'articulent entre eux, soulignant l'importance de les analyser dans une même trame. Les chapitres suivants développent un état de l'art des problèmes ainsi définis, identifient les principales approches et mettent en lumière les problèmes de recherche encore ouverts.

1.2 TAXONOMIE

La surveillance automatique, le nettoyage de zones sécurisées, la patrouille de frontière, le suivi ou la poursuite de cible sont autant de scénarios typiques et relatifs aux cibles en robotique mobile. Dans la plupart des cas, l'environnement est globalement connu : les tâches d'exploration ne sont pas abordées dans cette analyse. Les cibles considérées peuvent être mobiles ou fixes, mais notre étude se concentre principalement sur les cibles mobiles, présentant plus de difficultés.

La taxonomie est résumée par la figure 2 : elle se présente comme un arbre pour lequel chaque branchement est défini par un critère spécifique et lié à la définition d'un problème. Chaque feuille renvoie à une classe typique de problèmes, chacune incluant des variations dans leur formulation ou les hypothèses de départ.

Notre taxonomie définit également un vocabulaire cohérent, que nous voulons transverse et commun aux différentes communautés de recherche. Nous avons essayé, autant que possible, de rester fidèles au vocabulaire déjà en usage, mais il peut rester quelques conflits, inévitables à cause des différences de notations et des champs lexicaux propres à certains auteurs. En effet, il n'existe pas de standard officiel – ni même *de facto* – pour les noms et définitions des différents problèmes rencontrés. Cela est particulièrement sensible lors du passage d'une communauté à une autre : le même mot peut désigner des problèmes bien distincts dans la littérature, et c'est pourquoi nous avons choisi, par souci de clarté et de cohérence, de nous en tenir à notre taxonomie et au vocabulaire qu'elle introduit pour toute la suite de notre analyse.

Le premier critère de branchement de la taxonomie renvoie aux connaissances préalables sur les cibles et leurs positions. Cela définit deux grandes

classes de problèmes qui, en pratique, se succèdent généralement au sein d'une même mission : détecter des cibles (lorsque leurs positions sont inconnues ou incertaines), et pister ces cibles, lorsque leurs positions initiales sont établies.

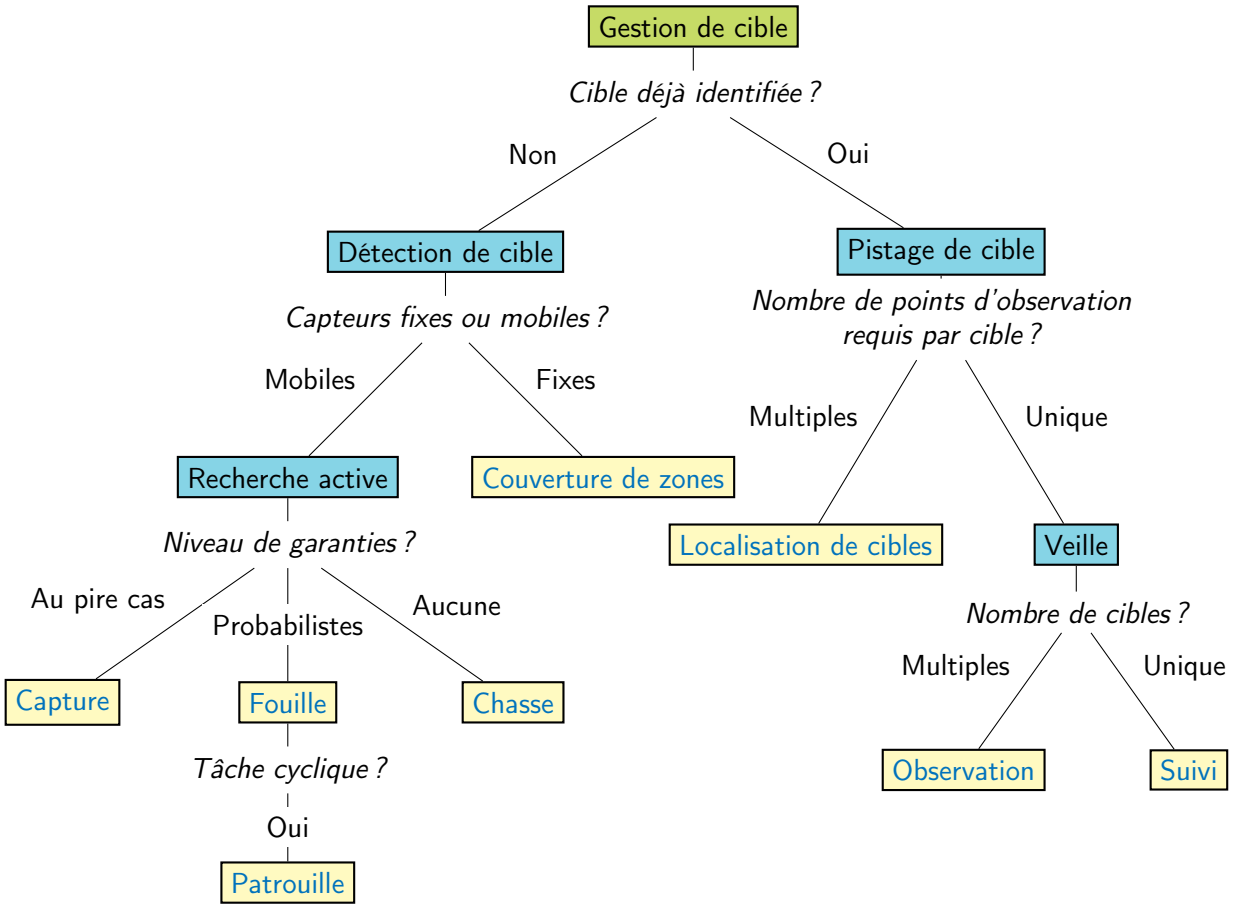


FIGURE 2 – Taxonomie des problèmes de gestions de cibles : les embranchements correspondent aux critères de la taxonomie, définis par des questions ; les feuilles (en jaune pâle) correspondent chacune à une classe spécifique de problèmes, définie dans les sections suivantes. Ces classes sont détaillées et analysées dans les chapitres suivants.

1.2.1 *Détection de cibles*

La *détection de cibles* consiste à trouver – « détecter » – une cible dans un environnement donné. Il peut y avoir une ou plusieurs cibles, et le nombre de capteurs (robots) disponibles n'est pas un critère à ce niveau. Ceux-ci peuvent en outre balayer activement l'environnement grâce à leur mobilité ou bien rester à une position fixe et surveiller passivement l'environnement.

On désigne cette dernière classe de problèmes par l'expression couverture de zones : les problèmes rassemblés dans cette catégorie se concentrent sur le placement des capteurs, ce qui implique souvent de partitionner l'environnement puis de distribuer les capteurs selon cette partition.

Lorsque la mission exploite explicitement la mobilité des capteurs, la planification des déplacements se retrouve au cœur des stratégies proposées. On parle alors de *recherche de cibles*. De tels problèmes peuvent être résolus localement ou globalement. Suivant les modèles utilisés et les hypothèses de départ, les stratégies formulées fournissent des garanties dans le pire cas (capture), des garanties probabilistes (fouille) ou aucune garantie (chasse). La recherche de cible peut être cyclique (patrouille), bien que cela ne présente pas d'intérêt dans le cas d'une capture ou d'une chasse. On notera que, pour chaque problème de *recherche de cible*, on peut définir une variante nécessitant d'entourer la cible pour éviter qu'elle ne s'échappe, plutôt que de simplement la détecter avec les capteurs ou de « l'attraper » en l'atteignant.

Les problèmes de détection décrits ci-dessous sont plus largement développés dans le chapitre 2.

COUVERTURE DE ZONES L'objectif d'une couverture de zones est de déterminer les positions optimales d'un ensemble de capteurs fixes afin de surveiller une zone définie. La forme traditionnelle de la couverture de zones est le célèbre *problème des gardiens de musée* (*the Art Gallery problem*), pour lequel il existe de nombreux travaux et résultats – voir la section 2.1 pour plus de détails. Certaines variantes de la couverture de zones considèrent des capteurs mobiles, mais les stratégies obtenues se concentrent toujours sur le positionnement des capteurs – « *où* placer les capteurs ? » – plutôt que sur les déplacements de ces capteurs – « *comment* atteindre les positions retenues ? ».

CAPTURE L'optimalité et la complétude sont des caractéristiques essentielles du problème de capture : le but est de « nettoyer » une zone connue et délimitée tout en fournissant des garanties au pire cas, ce qui signifie que toute cible se trouvant dans la zone définie sera trouvée quelles que soient ses capacités*. Pour cela, l'équipe poursuivante essaie d'encercler les cibles présentes. Aucune hypothèse sur ces dernières n'est formulée : leurs positions ne sont pas estimées et les cibles peuvent même avoir des capacités extraordinaires comme une vitesse infinie.

On peut établir un lien entre les problèmes de capture et le fonctionnement des battues à la chasse.

Les problèmes de capture sont aussi désignés sous d'autres termes, parmi lesquels « poursuite / évasion » (*pursuit-evasion*), « sécuriser » (*search and secure*) ou encore le jeu « des gendarmes et des voleurs » (*cops and robbers* games).

Les travaux dans cette classe de problème sont souvent basés sur des résultats mathématiques solides, et l'on peut distinguer deux principales approches [10, 41] : poser le problème comme étant celui d'un « nettoyage de graphe » (*graph clearing*), ou l'aborder d'un point de vue purement géométrique, dans un environnement plan et polygonal. Une variante courante consiste à trouver le nombre minimal d'agents nécessaires pour obtenir des garanties au pire cas. Ces approches sont discutées plus en détails dans la section 2.2.

FOUILLE La principale différence entre les problèmes de capture et les problèmes de fouille porte sur les garanties au pire cas, absentes pour les seconds. En effet, ces derniers sont caractérisés par des considérations probabilistes, et fournissent des garanties en accord [163]. Ce type de considérations est principalement motivé par un manque de ressources – manque de robots ou de temps – qui ne permet pas de gérer les pires cas. Cela peut également être motivé par la recherche d'un compromis entre l'efficacité et la probabilité d'apparition de certaines situations particulières et difficiles.

Les solutions aux problèmes de fouille exploitent les probabilités de distributions connues ou estimées *a priori*, définies sur les modèles d'environnement – la position estimée des cibles par exemple. La plupart des auteurs tentent de borner la probabilité de détection des cibles. Celles-ci peuvent être antagonistes ou non, ce dernier cas étant plus simple à gérer car induisant une complexité algorithmique bien plus faible. Ce modèle de cible est celui utilisé dans les scénarios de « secours » (*search and rescue*), pour lesquels l'urgence de la situation empêche de mener une recherche exhaustive des victimes dans la zone et impose donc des priorités, ce que gèrent très bien les modèles probabilistes. La section 2.3 fournit une description étendue des problèmes de fouille et des algorithmes associés.

PATROUILLE Lors que la recherche active de cible – avec capteurs mobiles – se répète dans le temps, on emploie le terme de problèmes de patrouille. La patrouille est comme une version cyclique de la fouille, qui implique une analyse statistique des performances au cours du temps. Elle considère notamment le temps écoulé entre deux observations ou « visites » d'une même position. C'est un domaine de recherche plutôt récent, qui s'est réellement développé au cours de la dernière décennie [137]. Les travaux relatifs à ces problèmes sont présentés dans la section 2.4.

CHASSE Enfin, il existe des situations pour lesquelles aucune garantie n'existe quant à la détection ou la capture des cibles. Nous désignons cette classe de problèmes comme étant des problèmes de chasse. Cette absence de garantie vient d'un manque de ressources – robots ou temps – mais aussi d'un manque d'informations, en l'absence desquelles on ne peut fournir ni exploiter de modèles probabilistes sur la position des cibles. La chasse se place par ailleurs souvent dans un contexte multirobot ; les travaux correspondant sont présentés en section 2.5.

1.2.2 *Pistage de cibles*

La seconde grande classe de problèmes, le « pistage de cible », désigne les tâches qui surviennent lorsqu'une ou plusieurs cibles ont été détectées ou désignées – et succède donc généralement à une tâche de détection réussie. Dans le cadre du pistage, gérer une cible peut renvoyer au fait de l'attraper ou de la garder en vue afin de fournir des informations additionnelles – essentiellement sa localisation précise au cours de temps mais l'identification

d'une cible est aussi un objectif. Dans tous les cas, les pisteurs doivent rester « proches » de la cible, la distance requise descendant à zéro lorsque l'on souhaite l'attraper.

Plusieurs robots peuvent être nécessaires pour gérer une seule cible suivant les contextes. Il est par exemple préférable d'utiliser des points de vue multiples sur une cible pour définir précisément sa position : ce type de problèmes est défini comme étant de la localisation de cibles. Lorsqu'un seul point de vue est suffisant, on peut aussi distinguer les problèmes monocibles / monorobots (suivi) des problèmes multicibles / multirobots (observation).

Les travaux relatifs aux problèmes de pistages décrits ci-après sont plus largement développés dans le chapitre 3.

LOCALISATION DE CIBLES Les problèmes de localisation de cibles consistent à traquer une cible à l'aide de plusieurs capteurs afin d'augmenter les connaissances sur la cible en question, notamment la précision de sa position estimée. Ce sont donc fondamentalement des problèmes multirobots dont la solution consiste à déterminer différents points de vue pour maximiser le gain d'information. La fusion de données, la coopération multirobot, les communications et la localisation multirobot sont au cœur de ce type de problèmes. Il est possible de considérer plusieurs cibles à la fois, chacune nécessitant d'être observée depuis plusieurs points de vue différents – voir la section 3.1.

SUIVI Les problèmes de suivi sont très couramment appelés « problèmes de poursuite-évasion ». Le lecteur intéressé pourra trouver dans [6] un bref historique des travaux dans le domaine, et se réfèrera aux travaux de Nahin pour un historique plus complet et une étude approfondie du sujet [119]. Les premiers travaux sur le suivi se rapportent aux conflits navals, et la forme traditionnelle du problème de poursuite en mathématique est connue sous le terme de « problèmes de l'homme et du lion » (*Lion and Man*). Dans sa version d'origine, un poursuivant unique – le lion – cherche à attraper une proie unique – l'homme – en se déplaçant à la même vitesse. Depuis, de nombreuses versions du problème ont été étudiées, avec des variations dans les vitesses respectives, les formes d'environnements (obstacles y compris), les conditions de visibilité, *etc.* La classe des problèmes de suivi rassemble toutes ces variations, qui sont des problèmes de pistage de cible impliquant une seule cible et un seul poursuivant – voir la section 3.2.

OBSERVATION On définit les problèmes d'observation comme suit : étant donnés une équipe de plusieurs robots et plusieurs cibles mobiles désignées, comment déplacer les robots pour observer simultanément l'ensemble des cibles, ou à défaut minimiser le temps pendant lequel une cible n'est pas observée par au moins un robot ? Ce problème a été rigoureusement décrit par Parker en 1997 [128], qui s'y réfère sous l'acronyme anglais *CMOMMT* (pour *Cooperative Multi-robot Observation of Multiple Mo-*

ving Targets, soit « observation de cibles mobiles par une équipe de robots coopératifs »).

L'une des principales difficultés de cette classe de problèmes consiste à répartir correctement les cibles entre les robots, et à décider quand et comment les observateurs devront s'échanger entre eux les cibles dont ils ont la charge (se référer à la section 3.3 pour plus de détails).

1.3 État de l'art

À partir de cette taxonomie, nous présentons un état de l'art de chacune des huit classes de problèmes décrites. Les chapitres 2 et 3 développent respectivement les problèmes de détection de cibles et ceux de pistages de cibles. Chacune de leur section présente l'état de l'art d'une classe de problèmes, et elles peuvent être lues indépendamment les unes des autres – notamment, chacune comporte un résumé qui peut en faciliter ou accélérer la lecture. Le chapitre 4 s'appuie sur ces développements et présente une synthèse critique des approches et modèles rencontrés. Cette analyse fonde la suite de nos travaux présentés en partie II, et son contenu peut être lu indépendamment des chapitres 2 et 3.

2

DÉTECTION DE CIBLES

Un peuple qui oublie son passé se condamne à le revivre.

— attribué à Winston Churchill

Ce chapitre décrit plus en détails la partie gauche de notre taxonomie (figure 2 p. 7) : il rassemble les travaux remarquables ou originaux se rapportant à la détection de cibles. Nous ne cherchons pas à être exhaustifs : le lecteur intéressé pourra se reporter aux analyses existantes, spécifiques à certains problèmes [169, 41, 102, 137, 163]. Nous nous concentrons sur les principales approches et sur quelques travaux sortant de l'ordinaire. Nous mettons aussi en avant les limites de certaines approches, bien que cela soit développé plus en détails dans le chapitre 4.

Remarque. Ce chapitre est organisé en sections indépendantes les unes des autres, chacune correspondant à une classe de problèmes de détection de cibles définie dans la taxonomie. Chaque section se termine par un résumé ; ces derniers peuvent permettre une lecture plus rapide de ce chapitre.

2.1 COUVERTURE DE ZONES – GARDIENS DE MUSÉE

Étant donnés une zone connue et un ensemble de cibles fixes, nous définissons les problèmes de couverture de zones comme la recherche d'une répartition optimale de ces capteurs, positionnés dans le but de couvrir entièrement la zone tout en minimisant le nombre de capteurs requis. C'est une tâche centrale au cœur de nombreuses applications pratiques, dont les scénarios de surveillance. Ce type de problèmes est aussi désigné sous le terme de problème des gardiens de musée. Cette section en présente différentes variations et les travaux associés.

GARDIENS DE MUSÉE

Le fameux problèmes des gardiens de musée (*art gallery problem*) est initialement proposé par Victor Klee en 1973 en réponse à Vasek Chvátal demandant un problème de géométrie intéressant [123]. Dans sa formulation d'origine, le but du problème est de déterminer le nombre minimal de gardiens nécessaires ou suffisants pour couvrir l'intérieur d'une galerie de musée polygonale à n murs – voir figure 3. Chvátal a d'abord prouvé que $\lfloor n/3 \rfloor$ gardiens sont parfois nécessaires et toujours suffisants pour surveiller une gallerie de forme polygonale simple à n côtés. Ce problème a été intensivement étudié depuis, avec de nombreuses variations dans la forme des environnements ou l'ajout de contraintes sur les gardes, leurs positions ou leurs capacités de « vision ».

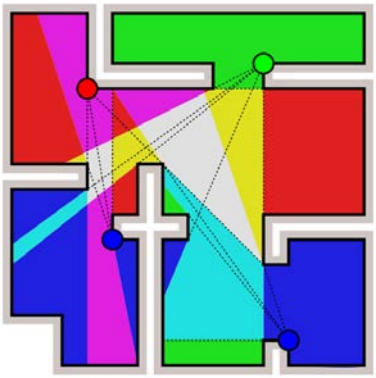


FIGURE 3 – Un exemple de solution au problème des gardiens de musée, avec quatre gardes – tiré de Wikipedia.

Sous sa forme générale, il a été démontré que le problème des gardiens de musée est NP-difficile [123]. De nos jours, l'état de l'art fournit de nombreux résultats sur le nombre minimal de gardes requis, mais il est toujours complexe de trouver les positions associées. L'excellente étude d'Urrutia sur le problème [169] rassemble la majeure partie des résultats de l'état de l'art, incluant les multiples variantes. Cette analyse présente aussi le problème des gardes mobiles et développe les relations entre ce nouveau problème et le problème original, qui expliquent d'ailleurs pourquoi les principaux résultats sur les gardiens mobiles concernent le nombre minimal de gardiens plutôt que les chemins empruntés par ces gardiens. Pour ces derniers, le lecteur se référera plutôt aux sections 2.2 et 3.2.

Le problème des gardiens de musée étant central pour de nombreuses applications d'ingénierie, connaître le nombre minimal de gardiens ne suffit pas : il est nécessaire d'avoir des stratégies de positionnement effectives. Parmi eux, Gonzales et Latombe approximent le problème des gardiens de musée pour en donner une solution pratique [65]. Leur méthode fonctionne à partir d'un échantillonnage transformant le problème d'origine en un problème de couverture par ensemble. Ils justifient leur approche par la complexité du problème et par la notion d' « élasticité » de l'optimal : la plupart des positions optimales peuvent être modifiées légèrement sans changement notable sur la qualité de la solution. Et lorsque celle-ci est détériorée, la solution considérée n'est sans doute pas souhaitable en pratique à cause de son manque de robustesse. Le lecteur souhaitant approfondir ces techniques par approximation peut se référer aux travaux de Ghosh [62].

VARIATIONS Comme expliqué plus haut, la version historique du problème de couverture de zones est le problème des gardiens de musée, mais de nombreuses variations ont vu le jour depuis l'apparition du problème original. Certains des travaux rassemblés dans cette section utilisent des capteurs mobiles, mais comme indiqué au chapitre 1 présentant la taxonomie, les approches se concentrent malgré tout sur les positions et non sur les chemins, d'où leur place dans cette section.

Ainsi Rybski utilise-t-il des robots mobiles pour aborder le problème de couverture de zones : il considère une combinaison de scouts et de rangers [154], ce qui explique pourquoi son approche ne se base pas sur les résultats classiques du problème des gardiens de musée. Scouts et rangers sont tous deux des capteurs mobiles : le ranger est une plateforme de grande taille, capable d'une grande autonomie énergétique et servant de base mobile à un ensemble de scouts, qui sont des petits robots bon marché déployés et supervisés par le ranger. Une fois déployés, l'objectifs des scouts est de prendre position dans des zones sombres où ils peuvent se cacher et détecter toute intrusion. Ces zones sombres sont détectées localement, et les scouts sont capables de s'adapter dynamiquement aux changements de luminosité. Le système cherche avant tout des positions adéquates pour les scouts ; le ranger peut être amené à se déplacer pour récupérer ou partager des données avec ses scouts.

De leur côté, Kloder et Hutchinson définissent et étudient le problème de *couverture de frontière* (*Barrier Coverage*) [85]. Celui-ci est très similaire au problème des gardiens de musée, le but étant de couvrir non pas des zones mais leurs frontières afin de détecter toute intrusion. Les environnements considérés sont plans, bornés et polygonaux, et les frontières à surveiller sont des segments. Le problème est résolu en trouvant une frontière de taille minimale : pour cela, les auteurs construisent un graphe de frontières candidates et réduisent le problème à celui d'un flot-max/coupe-min, bien connu dans la littérature sur les graphes. Nous trouvons cette approche originale et intéressante : sous l'hypothèse qu'un intrus ne puisse pas surgir n'importe où dans la zone à couvrir, la couverture de frontières permet de réduire de nombre minimal de gardes nécessaires par rapport à un problème de gardiens de musée. Néanmoins, les solutions ne tiennent pas face à des menaces telles que le parachutage, et ne sont pas adéquates pour la surveillance de zones, par exemple pour détecter les départs de feux de forêt.

Pimenta et collab. abordent eux le problème du *Simultaneous Coverage And Tracking* (SCAT) [133]. Ils utilisent également des capteurs mobiles, dont les positions stationnaires sont déterminées par un diagramme de Voronoï adapté au nombre de robots. Lorsque des cibles sont détectées, elles sont attribuées à un robot spécifié par la partition ; les positions des cibles influencent celles des robots en modifiant la fonction de densité sur laquelle se base la décomposition. De cette manière, l'équipe de robots accomplit à la fois un problème de couverture de zones et un problème de pistage, plus précisément un problème d'observation – voir aussi la section 3.3. Cette approche inhabituelle met en évidence les liens entre la partie gauche (détection) et la partie droite (pistage) de notre taxonomie. Les auteurs fournissent un ensemble de résultats expérimentaux basés sur des simulations *ad hoc*, des simulations avec Gazebo* et des tests embarqués sur robots. Il apparait cependant qu'aucun des environnements testés ne contient d'obstacle, ce qui limite fortement la portée des résultats présen-

Gazebo est un simulateur de robots réaliste – voir `http://gazebosim.org/`

tés, obstacles et environnements encombrés étant naturellement sources de nombreuses difficultés.

À cause de la configuration du terrain ou d'un manque de capteurs disponibles, il peut rester des trous dans la couverture d'une zone par des capteurs fixes. En outre, il peut être intéressant d'ajouter temporairement un angle de vue supplémentaire sur une cible donnée, cela étant coûteux dans le cas permanent sur l'ensemble de la zone à couvrir. Face à ce constat, Shi et collab. proposent l'utilisation d'un réseau hybride de capteurs fixes et mobiles [160]. Ces derniers sont utilisés pour « patrouiller » entre les trous de la couverture des capteurs fixes, et apportent des angles de vue supplémentaires lorsque cela est demandé.

Une variété riemannienne est une variété différentielle munie d'une métrique permettant de définir la longueur d'un chemin entre deux points de la variété.

Plus récemment, Bhattacharya et collab. se sont intéressés aux environnements complexes pour lesquels l'usuelle distance euclidienne n'est pas vraiment adaptée [27]. Ils donnent d'intéressants résultats valables pour toute variété riemannienne* bornée. Leur résultats théoriques sont valables dans les espaces continus, mais ils soulignent eux-mêmes le fait que face à la charge de calcul induite, la discrétisation est un « compromis indispensable ».

RÉSOLUTION PAR OPTIMISATION Si le problème des gardiens de musée est très proche du problème de couverture par ensemble, il peut également être considéré comme un problème d'optimisation. De manière plus générale, c'est le cas de tout problème de couverture de zones. En effet, les techniques d'optimisation sont particulièrement adaptées aux problèmes dont la solution est numérique et fixe : un ensemble de positions rentre parfaitement dans ce cadre. On pourra par exemple se référer aux travaux de Renzaglia [146], qui utilise une technique d'optimisation adaptative appelée CAO (pour *Cognitive-based Adaptative Optimization*), afin de résoudre le problème de couverture de zones pour une flotte de drones. Bien que les techniques d'optimisation soient avant tout centralisées, l'auteur propose une variante décentralisée de son algorithme, tout comme une extension au problème en trois dimensions.

RÉSUMÉ Le problème de couverture de zones est un problème bien connu et largement étudié. Il a fait l'objet de nombreux travaux depuis sa formulation originale, le problème des gardiens de musée. Les approches proposées sont plutôt théoriques et formelles, avec une forte implication de la communauté mathématique, et les principaux résultats concernent le nombre de gardes nécessaires plutôt que les stratégies de placement. Plus récemment, des variations ont introduit des capteurs mobiles en soutien des capteurs fixes. Malgré l'étendue des résultats théoriques, l'état de l'art du problème de couverture de zones manque de résultats pratiques, et l'on peut regretter l'absence générale de modèles riches ou réalistes, notamment en ce qui concerne les capteurs et les communications. Hors, ces derniers sont cruciaux dans l'efficacité des approches en contexte opérationnel.

2.2 CAPTURE – NETTOYAGE DE GRAPHES CONTAMINÉS

L'objectif du problème de capture tel que défini par notre taxonomie est de sécuriser, ou « nettoyer », une zone donnée, en capturant toutes les cibles se trouvant à l'intérieur. Les positions des cibles sont inconnues et, suivant les spécifications de la mission, il peut être nécessaire d'encercler une cible avec suffisamment de nettoyeurs pour la capturer ; autrement, détecter la cible suffit pour la neutraliser. Comme expliqué au chapitre 1, les notions d'optimalité et de garanties au pire cas sont fondamentales dans la définition des problèmes de capture.

TRAVAUX PIONNIERS En 1978, Torrence D. Parsons publia *Pursuit-Evasion in a Graph* [129] : il imagine alors qu'« un homme est perdu dans un réseau de cavernes, errant de manière imprédictible », tandis qu'« un groupe de secouristes connaissant la structure de la caverne est envoyé à sa recherche ». Il se demande quel est le nombre minimal de secouristes nécessaires pour retrouver l'homme, quels que soient ses déplacements. Notre définition du problème de capture hérite directement de cet énoncé : l'environnement est connu, les chercheurs ne connaissent pas la position de la cible, et des garanties au pire cas sont attendues, les déplacements de la cible étant imprédictibles. Le problème pour Parsons est alors de définir le nombre minimal de chercheurs nécessaires pour mener à bien la mission, plutôt que d'élaborer explicitement une stratégie de recherche. On désigne ce nombre minimal comme étant l'*encerclement* d'un graphe, aussi appelé *search number* [29].

Parsons décrit la structure de l'environnement comme celle d'un graphe ou d'un arbre. Il propose différents théorèmes permettant de déterminer l'encerclement d'un graphe donné de manière récursive à partir de ses sous-graphes. Les résultats de Parsons se concentrent sur les structures d'arbres qui sont plus simples à gérer que les graphes génériques. Ce premier article introduit de nombreux résultats intéressants mais ne fournit aucune stratégie pour les agents nettoyeurs.

Les graphes à sécuriser sont dits « contaminés », et l'on cherche à les nettoyer. Plus largement, les zones contaminées sont celles pouvant abriter une cible, tandis que les zones décontaminées ou nettoyées sont celles pour lesquelles on peut garantir qu'aucune cible ne s'y trouve. Il a été démontré depuis que déterminer l'encerclement d'un graphe est un problème NP-difficile dans le cas général, et NP-complet pour les graphes dont les sommets sont au plus de degré* 3 [100]. En outre, LaPaugh a démontré qu'autoriser la « recontamination » du graphe n'aidait pas à sa décontamination finale [93]. La recontamination d'une zone est le fait d'autoriser une zone précédemment nettoyée à être envahie à nouveau par une cible mobile, par exemple en ne gardant pas ses frontières. Le résultat de LaPaugh est crucial car il permet de juger *a priori* de la qualité de certaines solutions, et oriente la recherche de stratégies de nettoyage efficaces en décrivant leur forme générale.

Le degré (ou valence) d'un sommet d'un graphe est le nombre de liens (arêtes ou arcs) reliant ce sommet (les boucles comptant double).

GRAPHES CONTAMINÉS – JEU DES GENDARMES ET DES VOLEURS À la suite de ces travaux pionniers, de nombreux chercheurs ont étudié le nettoyage de graphes contaminés (ou *graph clearing*), qui est un cas particulier de problème de capture dans lequel l'environnement est décrit par un graphe, représentation discrète et abstraite à laquelle sont rattachés de nombreux résultats et algorithmes. On compte plusieurs variations de nettoyage de graphes contaminés, parmi lesquelles la recherche par arêtes (*edge-search*, aussi appelée *graph sweeping*), la recherche par sommets (*node-search*, qui est le cas traditionnel du nettoyage de graphes), la recherche de cibles antagonistes à vitesse infinie, *etc.* Le problème est aussi connu comme le *jeu des gendarmes et des voleurs* (*cops and robbers game*) [41]. Il peut spécifier que la position des cibles est connue – voir figure 4. Le lecteur souhaitant approfondir le sujet pourra se reporter à l'étude d'Alspach [10].

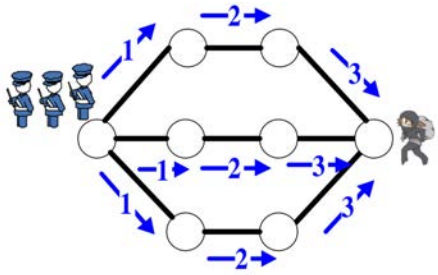


FIGURE 4 – Un exemple de jeu des gendarmes et des voleurs – tiré de [172].

Comme expliqué par Moors [106], la formulation du problème de capture dans un graphe soulève la question de la transformation d'un environnement géométrique en une structure topologique de graphe. Cette difficulté peut être surmontée par l'emploi d'algorithmes proches du problème des gardiens de musée. Il est également possible d'utiliser des « bloqueurs », c'est-à-dire de « sacrifier » des agents comme de simples gardes barrières – provisoirement stationnaires, afin d'éviter toute recontamination des zones nettoyées. Ce concept est particulièrement utile aux approches de type « diviser pour régner » et permet de réduire la complexité de certaines zones ou structures.

Dans un graphe, les longueurs désignent habituellement les longueurs des arcs. La largeur d'un arc ou d'un sommet désigne ici le nombre d'agents nécessaires respectivement pour nettoyer un arc ou garder un sommet.

Borie et collab. ont étudié des graphes aux structures particulières pour lesquels il existe des algorithmes en temps polynomial déterminant l'encerclement du graphe [31]. Il est ainsi possible de déterminer linéairement l'encerclement d'un arbre, d'un graphe intervalle, ou d'un graphe grille. Les auteurs fournissent par ailleurs un intéressant tableau résumant les résultats de complexité connus pour toute une variété de types de graphe, cela pour les problèmes d'encerclement, de distance minimale et de temps minimal. Ils proposent également un algorithme optimal permettant de déterminer l'encerclement d'un arbre, et étendent leur résultat aux graphes largeur-unitaire / longueurs-arbitraires* et longeur-unitaire / largeurs-arbitraires. Ils

montrent enfin le caractère NP-difficile du problème pour de nombreuses autres topologies de graphe.

Les travaux de Borie illustrent à quel point les considérations topologiques sont utiles pour réduire la complexité et fournir des solutions efficaces, voire optimales. C'est également le cas des travaux de Daniel qui utilise des graphes parallèles en série pour développer une approche heuristique valable même à grande échelle [45], la complexité de celle-ci étant de $O(nr^6)$ où n est le nombre de sommets dans le graphe et r le nombre de robots.

Un arbre couvrant d'un graphe non orienté et connexe est un arbre inclus dans ce graphe et qui connecte tous les sommets du graphe.

Quand le graphe ne présente pas de topologie particulière, c'est-à-dire qu'il n'appartient pas à une classe de graphes pour lesquels des algorithmes efficaces sont connus, il est toujours possible de modifier le graphe d'origine afin de l'« améliorer » notamment en bloquant (gardant) des arêtes ou des sommets. Hollinger et collab. utilisent par exemple des gardes pour transformer un graphe en arbre et résoudre un problème linéaire plutôt que NP-difficile dans le cas général [71]. Ils utilisent pour cela la notion d'arbre couvrant*. Leur algorithme est *anytime** et s'accompagne de garanties pour certaines de ses variations. Il est par ailleurs intéressant de noter qu'ils utilisent peu de communications entre robots, ces derniers se coordonnant partiellement de manière implicite. Dans le même ordre d'idée, Katsilieris et collab. soulignent également l'importance de garder certains sommets. Ils utilisent des robots stationnaires pour transformer un graphe en arbre en supprimant les boucles [84]. Ils analysent aussi le compromis entre le temps nécessaire et le nombre de robots utilisés pour nettoyer un environnement donné.

Un algorithme anytime *est un algorithme capable de donner une solution valide à un problème même s'il est interrompu avant d'avoir terminé. L'algorithme trouve de meilleures solutions au fur et à mesure de son exécution.*

La plupart des algorithmes utilisés pour le nettoyage de graphes contaminés font du parcours d'arêtes (*edge-searching*), dans lequel les agents se déplacent le long des arêtes et gardent les sommets. Kolling et Carpin remarquent que ce type d'algorithmes est difficile à implémenter en pratique [90]. En effet, dans cette configuration, un sommet correspond à deux actions distinctes* : détecter tout intrus dans la zone associée (c'est-à-dire nettoyer le sommet), ou empêcher toute recontamination, ce qui veut dire garder les entrées de la zone associée. Ils proposent donc un nouvel algorithme, *Graph-Clear*, dans lequel un robot peut soit bloquer une arête, soit nettoyer un sommet dont toutes les arêtes sont bloquées. De cette manière le nettoyage d'une zone donnée et la prévention de sa recontamination par les zones adjacentes sont clairement séparées, la première utilisant les nœuds et la seconde les arêtes. *Graph-Clear* est NP-difficile dans le cas général, mais il est possible de calculer une stratégie valide pour les arbres de manière quadratique en le nombre de sommets.

"Guarding a vertex performs two basic functions, namely, the prevention of spread of contamination from and to its neighbors and the detection of all intruders in the vertex". – Kolling and Carpin [90]

Il existe d'autres variantes du problème de nettoyage de graphes contaminés, portant sur les connaissances des agents ou sur les environnements modélisés par le graphe. Ainsi, Kolling considère explicitement la visibilité dans un environnement 2.5D [88] alors que les environnements habituellement considérés sont en deux dimensions. Cette demi-dimension supplémentaire est particulièrement adaptée aux environnements naturels. L'approche de Kolling se base sur un échantillonnage de l'environnement permettant de

construire le graphe désiré. Il suffit ensuite d'appliquer sur ce graphe un des algorithmes existants.

Vieira et collab. considèrent quant à eux le problème sous l'angle de la théorie des jeux, dans lequel les joueurs (cibles et nettoyeurs) sont chacun pleinement conscients de l'état des autres agents [172] – c'est un jeu à information complète. L'ensemble des protagonistes utilisent des stratégies optimales : pour ce faire, un diagramme de transition est calculé hors-ligne, et seul le choix des transitions locales est fait en ligne. Le calcul des stratégies optimales est exponentiel en le nombre de joueurs.

Le lecteur désirant encore approfondir le sujet des graphes contaminés peut se reporter à l'état de l'art des articles [106, 88] et à la section *cops-and-robbers* de l'étude de Chung [41].

APPROCHES GÉOMÉTRIQUES La formulation du problème de capture dans un graphe est intrinsèquement abstraite, presque « éthérée ». D'autres représentations permettent de se rapprocher du monde réel : elles placent la géométrie au cœur de l'approche et utilisent une représentation continue de l'environnement. Ce type de représentations permet d'utiliser des modèles de capteurs plus poussés et donne directement les détails d'une stratégie car elle est beaucoup plus « bas niveau* » que l'abstraction des graphes. Les environnements ici sont typiquement plans, polygonaux et contiennent des obstacles. Les capteurs sont explicitement considérés, généralement basés sur la vision (caméras, LIDARs), et les obstacles sont habituellement les mêmes pour le mouvement et la visibilité. Les algorithmes se concentrent dans la plupart des cas sur un seul agent nettoyeur, bien que certains étendent les stratégies résultantes aux équipes de robots.

En robotique, la notion de « bas niveau » s'oppose à celle de « haut niveau » faisant abstraction des caractéristiques techniques des robots.

Dans ce contexte géométrique, la question « quel est le nombre d'agents requis pour nettoyer une zone ? » rappelle immédiatement le problème des gardiens de musée (voir la section 2.1) : l'étude d'Urrutia rassemble les nombreux résultats sur le sujet [169], et nous nous concentrons donc ici sur les stratégies de déplacement des robots.

Ainsi, Gerkey et collab. s'intéressent à l'impact d'un champ de vue limité sur les stratégies des robots [61], comme illustré par la figure 5. Ils introduisent la notion de ϕ-*searchers*, donc l'angle de vue est restreint à ϕ degrés. Ils se concentrent sur l'étude d'un seul ϕ-*searcher*, puis étendent leur approche au cas multirobot avec un algorithme centralisé. Durham reprend et étend le concept de ϕ-*searchers* à celui de (d, ϕ)-*searcher*, qui prend aussi en compte la profondeur du champ de vision, limité à une longueur d [49]. Il aborde également le problème multirobot, proposant une approche décentralisée cette fois. Il est intéressant de noter que l'approche ne nécessite pas explicitement que l'environnement soit connu au préalable, bien que cela soit implicitement supposé par l'hypothèse que les agents sont assez nombreux pour mener à bien la mission.

La proximité des problèmes de capture monorobots monocibles et des problèmes de suivi est également soulignée par l'utilisation confuse du terme générique de « poursuite-évasion ».

Le lecteur attentif aura remarqué que la formulation géométrique du problème de capture est proche du problème de suivi dans le cas monorobot*. Ce constat se retrouve aussi dans l'étude de Chung [41]. Nous tenons à

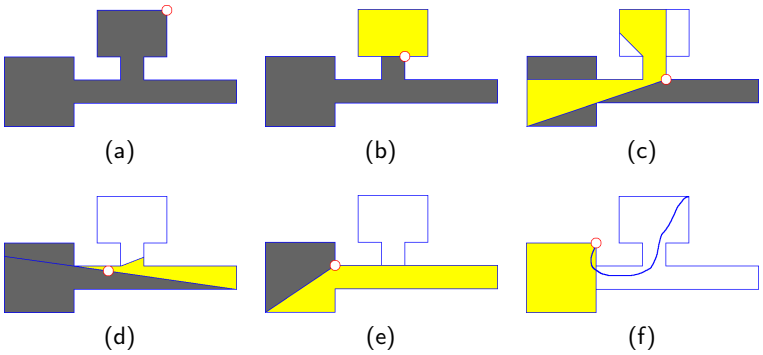


FIGURE 5 – Un exemple d'environnement nettoyé par un agent doté d'un champ de vue limité. Le nettoyage est effectué en sortant de la première pièce dos à la porte, puis en suivant le couloir jusqu'à la pièce de gauche – tiré de [61].

souligner une différence fondamentale malgré tout : ici le comportement des nettoyeurs est organisé autour de l'environnement, et la position de la cible est inconnue. En suivi, au contraire, la position de la cible est connue, celle-ci étant d'ailleurs initialement dans le champ de vision du poursuivant, et les mouvements du robot sont guidés par ceux de la cible – voir les sections 1.2 et 3.2.

"Planning in the joint configuration space of all searchers is clearly not the best approach, as it is centralized and scales badly as the number of searchers increases" – Gerkey [61]

COOPÉRATION DÉCENTRALISÉE Les travaux sur la capture présentés jusqu'ici ont un inconvénient majeur : les approches sont très majoritairement centralisées, ce qui introduit toutes les difficultés associées, y compris les problèmes de passage à l'échelle*. Les travaux de Durham et collab. évoqués précédemment [49] utilisent un schéma stratégique basé sur les frontières, à la manière des techniques d'exploration. Afin de résoudre un problème de capture et non d'exploration, les robots sont contraints dans les mouvements pour étendre les frontières existantes : ils ne peuvent laisser une frontière non surveillée. Les auteurs ont également introduit une notion de rôles dynamiques distinguant des *leaders* et des *followers*. C'est cette notion qui permet notamment de distribuer un algorithme centralisé à l'origine pour accomplir un nettoyage décentralisé.

*Les approches basées sur des enchères utilisent des mécanismes d'allocation de tâches s'inspirant des mécanismes d'enchères observés en salle des ventes ou en salle des marchés. On parle aussi de mécanismes de marché (*market-based*).*

De son côté, Tanner propose une approche partiellement décentralisée basée sur des algorithmes en essaim afin de contrôler une équipe hétérogène dans laquelle les AGVs et les AAVs ont des rôles distincts (respectivement de bloqueurs et nettoyeurs) [166].

Les algorithmes de coopération décentralisés sont particulièrement adaptés au problème de capture, et l'état de l'art contient plusieurs approches basées sur des enchères* (*auction-based*) comme solution au problème : Zhou considère une version étendue du fameux *Contract Net Protocol (CNP)* [180] tandis que Kalra propose Hoplites [79], dans lequel les robots nettoient une zone donnée en formant une ligne de balayage ; les positions

des robots dans celle-ci sont déterminés activement par un mécanisme de marché.

INTÉGRATION Comme indiqué en introduction de cette section, plusieurs algorithmes de l'état de l'art ont été intégrés à bord de robots et ont montré d'intéressantes performances, ce qui est assurément un signe de leur maturité. Kalra notamment a intégré son approche décentralisée par enchère dans des robots Pioneer [79]. Vieira a testé une équipe de robots iRobot Create couplés à un bâtiment intelligent soutenant leurs communications et fournissant des capacités de détection à distance [172]. Les capacités très restreintes des robots ont forcé Vieira à utiliser des mouvements par suivi de murs, mais cela s'est avéré suffisant pour des environnements de bureaux. La combinaison des bâtiments intelligents avec des robots bon marché – mais limités – semble prometteuse.

Le type d'environnement influence d'ailleurs grandement les approches et les performances observées, et l'on peut trouver d'autres exemples fonctionnant en environnement naturels simples [84] ou à l'intérieur de bâtiments [49].

Les travaux de Katsev se situent dans le cadre de la communauté « perception » [83], et son approche repose sur un robot suiveur de mur équipé de capteurs limités ne permettant pas une cartographie ni une localisation précises. Introduisant la notion de *zones d'ombre* (lorsqu'une cible peut se cacher derrière un obstacle), Katsev élabore une stratégie globale permettant de nettoyer l'ensemble de la carte, sous réserve qu'un seul robot suffise. L'algorithme est testé sur des plateformes Lego NXT. On peut regretter que l'algorithme ne soit pas complet, bien qu'il fournisse des garanties au pire cas lorsqu'il fonctionne.

RÉSUMÉ Parmi les nombreux travaux se rapportant aux problèmes de capture, il est possible de distinguer deux grandes tendances : une partie des approches raisonnent sur des graphes discrets, représentations abstraites de l'environnement, tandis que d'autres considèrent des représentations géométriques et continues. L'état de l'art des approches basées sur les graphes contient de nombreux résultats sur la complexité des algorithmes suivant la topologie des graphes considérés. Les approches basées géométrie s'accompagnent moins souvent de résultats sur leurs complexités, mais permettent de prendre en compte des modèles de capteurs beaucoup plus réalistes. Malheureusement, dans les deux cas, les approches restent très majoritairement centralisées. On observe cependant de plus en plus d'approches centralisées réellement efficaces, et quelques approches décentralisées. Les environnements considérés sont aussi de plus en plus variés et les modèles de capteurs complexes (visibilité 2.5D, champs de vue restreints). Enfin, plusieurs des approches de l'état de l'art ont été intégrées à bord de robots avec un certain succès, ce qui est indéniablement un indicateur de la maturité de ces approches et du savoir-faire dans le domaine de la capture.

2.3 FOUILLE PROBABILISTE

Les modèles probabilistes permettent de rendre compte des incertitudes et des priorités dans les problèmes de détection de cibles. Les travaux rassemblés dans cette section représentent les positions potentiellement occupées par une cible au travers de distributions de probabilités. Ce cadre probabiliste fournit certaines garanties de détection des cibles, sans pour autant assurer des performances au pire cas, ce qui se justifie généralement par un manque de ressources (temps, robots).

En outre, l'étude de l'état de l'art sur la fouille probabiliste montre que les approches multirobots sont la norme *de facto*. Les équipes de robots présentent de nombreux avantages tactiques sur un seul robot, mais soulèvent d'autres difficultés : passage à l'échelle, systèmes centralisés ou non, communications, *etc.*

L'état probabiliste des cibles est le plus souvent représenté par une distribution de probabilité discrète, reposant sur une grille cartésienne de l'environnement. Cette distribution évolue selon un modèle de cible connu et choisi. Les cibles peuvent en outre être antagonistes ou non. Avoir une cible indifférente aux agents la cherchant permet de réduire la complexité du problème à celle d'un jeu à un seul joueur tel que défini par Benkoski [23]. Ces modèles « indifférents » sont particulièrement adaptés aux problèmes de secours (*search and rescue*) [23, 41]. Les modèles markoviens sont les plus en vogue, mais ils conduisent à des solutions complexes. Des modèles de cibles statiques ou de type « marcheur aléatoire* » permettent de limiter drastiquement cette complexité car ils sont « stationnaires », au sens où ils ne dépendent pas des actions ou événements passés.

*Une marche aléatoire (*random walk*) est un modèle mathématique d'un système possédant une dynamique discrète composée d'une succession de pas aléatoires, ou effectués « au hasard », totalement décorrélés les uns des autres.*

La densité de la fonction de probabilité de présence d'une cible est par ailleurs un estimateur naturel de sa localisation : plusieurs travaux présentés ici cherchent ainsi à détecter des cibles puis à préciser leurs positions : ceci est alors proche des problèmes de localisation de cibles – voir la section 3.1. Bien que l'usage de distributions de probabilités nécessite une estimation *a priori*, la position des cibles n'est pas connue pour autant au sens décrit dans la taxonomie (voir le chapitre 1) : l'objectif premier des problèmes de fouille abordés ici est donc bien de trouver les cibles.

Le cadre probabiliste des problèmes de fouille offre en outre deux avantages : il permet de faciliter l'intégration à bord de vrais robots, en fournissant naturellement un moyen de gérer les incertitudes de la perception et des actions, et il permet de prioritiser les tâches entre elles. Ce dernier point est important lorsque l'aspect temporel est critique, comme en *search and rescue* où l'objectif est de trouver autant de cibles que possible en un temps très limité, plutôt que de trouver chacune des victimes, ce qui est peu réaliste et requiert sans doute beaucoup trop de temps par rapport à l'urgence de la situation. Face à des algorithmes exploitant ce type de préférence, on peut parler de « complétude probabiliste ».

En dépit du vocabulaire utilisé par certains auteurs, les problèmes de fouille sont bien distincts des problèmes de capture au sens défini par notre

taxonomie. Dans les travaux de Bopardikar par exemple [30], les robots ne sont pas assez nombreux pour fournir des garanties au pire cas comme en capture (du moins, il n'y a aucune intention de gérer les pires cas) : seules des garanties probabilistes sont disponibles. Malgré les déplacements méthodiques de balayage, les robots ne peuvent totalement empêcher la fuite d'une cible. Les bornes sur les probabilités de capture de la cible sont néanmoins importantes : elles permettent d'évaluer directement l'efficacité théorique des algorithmes.

TRAVAUX PIONNIERS Les premiers travaux en fouille probabiliste suivent l'idée qu'il n'est pas toujours possible de connaitre précisément la carte du lieu de la mission : dans ces cas là, utiliser une représentation par graphe, comme c'est souvent le cas en capture, n'est pas adapté. Hespanha et collab. considèrent un jeu de « poursuite-évasion » au sens de la détection de cible [68]. Plus précisément, ils considèrent des cibles de type marcheurs aléatoires dans un scénario proche des problèmes de *search and rescue*. Leurs robots utilisent une stratégie gloutonne, guidée par la maximisation de la probabilité conditionnelle *a posteriori* calculée à partir d'une carte quadrillée et imprécise donnant des probabilités de présence de cibles *a priori*. Cette carte évolue au cours du temps en fonction des déplacements et observations passés. Les déplacements et les capteurs ont aussi des modèles probabilistes, et le temps et l'espace sont tous les deux discrétisés. Les auteurs introduisent la notion importante de « stratégie persistante » (*"persistent policy"*), dans laquelle chaque déplacement garantit une chance non nulle de capturer la cible, c'est-à-dire une stratégie dans laquelle les mouvements prudents ont la préséance sur les mouvements qui « tentent le tout pour le tout. » (*no "make-or-break" move*). L'approche d'Hespanha et collab. ne fournit pas de solution optimale, mais s'appuie sur des fondements mathématiques solides et présente de nombreuses similitudes avec les travaux plus récents sur le sujet : modèles probabilistes, grilles, marcheurs aléatoires, stratégie persistante, *etc.* On peut néanmoins regretter que seuls des résultats issus de simulations *ad hoc* viennent soutenir ces travaux.

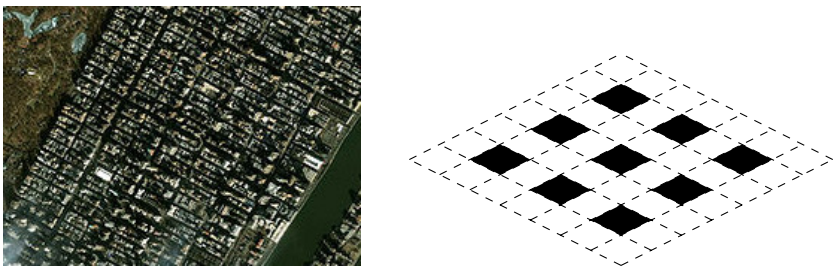
MODÈLES MARKOVIENS Les travaux pionniers d'Hespanha ont inspiré de nombreux autres travaux en fouille, et une grande partie d'entre eux utilisent des modèles markoviens. Hollinger et collab. considèrent ainsi la fouille d'un environnement connu par une équipe de robot à la recherche d'une cible mobile non antagoniste [69]. Ils démontrent que le problème de calcul de chemin résultant est NP-difficile, et que trouver la solution optimale est exponentiel en le nombre de robots chercheurs. Face à cette complexité, ils proposent des stratégies d'approximation dont les performances sont garanties, bornées par rapport à l'optimal. Plus précisément, ils considèrent une formulation POMDP sur l'union des espaces de positions des robots et de la cible, bien que cela limite fortement le nombre de protagonistes et la taille des environnements. Afin de prévenir ces limita-

tions, les robots se coordonnent de manière implicite : ils ne planifient que pour eux-mêmes, dans un horizon temporel* fini, et partagent ensuite leurs plans futurs et passés.

Un horizon temporel est la limite de temps future au-delà de laquelle plus aucune planification n'est considérée.

Les résultats de complexité fournis par Hollinger sont particulièrement intéressants. L'hypothèse que la carte de l'environnement est connue *a priori* permet en outre de borner assez facilement les performances du système. Nous pensons également que l'étape de modélisation est cruciale (voir aussi le chapitre 5 à ce sujet) : il est donc important de noter que les cartes et graphes représentant les environnements sont définis manuellement, bien que les auteurs affirment que cela puisse être automatisé. La plupart des résultats fournis sont par ailleurs issus de simulations *ad hoc*, le système n'ayant été que partiellement intégré à bord d'un seul robot Pionneer.

Les POMDPs sont un outil puissant souvent utilisé pour modéliser les mouvements de cibles. En fouille probabiliste, comme déclaré plus haut, de nombreux travaux utilisent une grille cartésienne comme représentation de l'environnement, car c'est une discrétisation aisée à manipuler. Les POMDPs sont alors utilisés pour décrire le modèle de comportement (de déplacement) des cibles d'une case à l'autre. Couplés aux observations, ils permettent de faire évoluer au cours du temps la distribution de probabilité de la présence des cibles. Les travaux de Yu et collab. sont un très bon exemple de ce type de représentation, couplés dans leur cas avec des modèles d'AAVs à voilure fixe rendant compte de la cinématique particulière de ces derniers [178]. Les environnements considérés sont de type « Manhattan » (figure 6) ; les possibilités de mouvements sont échantillonnées pour mieux gérer la complexité.



(a) (b)

FIGURE 6 – Exemple d'environnement de type « Manhattan » : (a) une vue satellite de Manhattan : on distingue nettement le quadrillage des rues ; (b) un exemple de modèle d'environnement de type Manhattan : un quadrillage régulier, avec des obstacles réguliers en noir et les espaces libres en blanc.

Il existe néanmoins des alternatives à cette représentation « canonique ». Toujours en utilisant des modèles markoviens, Ferrari considère par exemple le problème de fouille comme un problème d'optimisation géométrique [58], et cherche à maximiser la probabilité de détection tout en minimisant la consommation d'énergie des robots, et ce en considérant les contraintes de visibilité des robots chercheurs.

CIBLES ANTAGONISTES Comme expliqué au début de cette section, pour des raisons de complexité, la plupart des modèles markoviens décrits ci-dessus considèrent des cibles non antagonistes : non seulement cela permet de considérer des environnements plus larges et un nombre de protagonistes (cibles et robots) plus grand, mais cela permet aussi de faire des considérations plus globales. En effet, dans la plupart des cas les cibles non antagonistes « ignorent » tout simplement les robots, ce qui veut dire qu'elles ne coopèrent pas avec eux dans le but d'être retrouvées, mais leur comportement n'est pas non plus influencé par celui des robots. Il est alors possible de prédire l'évolution de la distribution de probabilité de présence des cibles à faible coût, éventuellement hors-ligne. Cette indépendance des cibles permet de simplifier grandement les équations où leur probabilité de mouvement et de présence apparaissent. Cette hypothèse d'indépendance est par ailleurs pleinement justifiée par certains scénarios concrets, comme ceux de *search and rescue*. Il existe néanmoins un espace à combler entre les cibles antagonistes souvent considérées en capture et les cibles indifférentes en fouille.

Strom et collab. font ce même constat et proposent de combler ce manque [164]. Pour cela, ils modélisent les mouvements des cibles antagonistes par des modèles semi-aléatoires et paramétriques. Ils utilisent un algorithme de parcours en profondeur sur un arbre élagué simulant le jeu de cache-cache ou de poursuite entre les chercheurs et les cibles. Le facteur de branchement de l'arbre d'états* croît exponentiellement avec le nombre de chercheurs et polynomialement seulement avec le nombre possible d'actions de l'ensemble des protagonistes. Inspirés par les travaux d'Hollinger [69], Strom aborde le problème comme une maximisation de récompenses (*rewards-maximizing search*) ; une planification séquentielle basée sur une coopération implicite entre les robots permet de linéariser le caractère exponentiel du facteur de branchement au prix de la perte de l'optimalité.

Un arbre d'états est constitué d'un état d'origine et de ses états successeurs possibles, chaque branche représentant une transition d'état, liée à une action précise ou un événement particulier.

Le caractère semi-aléatoire des cibles permet lui aussi de réduire la complexité : les cibles échantillonnent leurs actions possibles et choisissent les meilleures parmi ces échantillons. Le nombre d'échantillons est un paramètre permettant de définir l'habileté de la cible : un seul échantillon donne un marcheur aléatoire, alors qu'un échantillonnage exhaustif revient à considérer une cible pleinement antagoniste. Définir quelle est la meilleure option parmi l'échantillon obtenu n'est néanmoins pas trivial, et cet aspect n'est malheureusement pas vraiment discuté dans l'article.

D'après les résultats présentés, le système de Strom gère aussi bien les cibles aléatoires que les cibles pleinement antagonistes ou que les cibles avec une stratégie fixe. Cette polyvalence le rend très intéressant, mais il a plusieurs limites, soulignées ou non par ses auteurs. En particulier, il est nécessaire d'estimer quel est le niveau d'habileté des cibles, ce qui n'est pas évident. Les problèmes de communications ne sont pas abordés et les environnements sont considérés de manière très abstraite sous forme de graphe, ce qui peut poser des problèmes d'intégration.

La validation expérimentale n'est d'ailleurs pas convaincante : le système est testé à l'aide de trois robots à roues sur un parking, avec des cibles humaines. L'environnement est simple (grille sans obstacle) et surtout les agents ne sont autorisés à bouger que d'un seul segment à chaque pas de temps, ce qui veut dire que le processus expérimental intègre les mêmes hypothèses de départ que le système (discrétisation spatiale et temporelle). Comme nous le verrons à la section 4.3, cela cache *de facto* les problèmes associés à cette discrétisation et qui surviennent lors de la mise en application réelle.

FILTRAGE PARTICULAIRE Il est largement admis que les POMDPs sont un outil puissant dont la complexité est néanmoins problématique face à des problèmes déjà complexes en eux-mêmes [152]. Les filtres particulaires sont une technique d'estimation classique permettant de rendre compte d'une distribution de probabilité avec des capacités de calcul raisonnables. Ils sont le plus souvent utilisés pour la localisation d'un robot ou le pistage de cible, mais leur usage dans un problème de recherche de cible est immédiat. Ils présentent aussi l'avantage de n'utiliser qu'un seul et même algorithme pour trouver puis pister une cible, même si d'autres techniques de pistage peuvent être plus adaptées – se référer au chapitre 3.

Mottaghi et Vaugan utilisent un filtre particulaire dans un contexte de fouille multirobot coopérative [109]. Ils montrent comment la coordination permet d'améliorer les performances d'un système multirobot. Leurs robots échangent leurs positions, et, au choix, leurs observations ou les particles représentant les cibles, les deux représentant grossièrement la même information. Le schéma de coordination est implicite, chaque robot prenant en charge les particules dont il est le plus proche. L'approche résultante est donc linéaire avec le nombre de robots, et il suffit d'échanger quelques données : il n'y a pas de négociation nécessaire. Chaque robot est ensuite guidé par un champ de potentiel directement issu des particules dont il a la charge. Un algorithme de Dijkstra permet d'évaluer la force et la direction des forces en question, le but étant de maximiser le nombre de particules visibles, ce qui réduit l'entropie de la distribution.

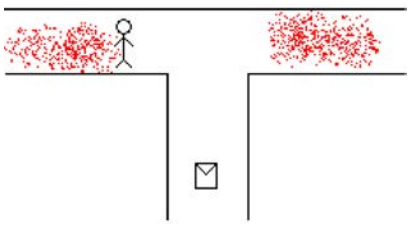


FIGURE 7 – Stratégie de fouille probabiliste à base de filtrage particulaire : le nuage de points rouges représente l'ensemble des hypothèses sur la position courante de la cible (= échantillons de la distribution de probabilité associée). Ces hypothèses, et donc la distribution associée, évoluent selon un modèle probabiliste de déplacement de la cible – tiré de [109]

La validation expérimentale de ce système implique des simulations *ad hoc* et des tests en environnement d'intérieur avec des robots Pioneer dont les résultats sont prometteurs. Les avantages et inconvénients de l'approche sont bien discutés par les auteurs : il est entre autres souligné qu'il n'y a pas de revendication d'optimalité mais que les comportements observés sont tout à fait satisfaisants. Il est également rappelé qu'à cause des minimaux locaux dans le champ de potentiel, il est possible que des boucles dynamiques se forment – ce problème est inhérent à tout champ de potentiel. De manière regrettable mais classique, le système présuppose l'existence d'un réseau de communications parfait entre les robots : absence de coupures et tous les robots sont liés les uns aux autres quelles que soient leurs positions. En revanche, le modèle d'environnement est continu, l'approche permet de gérer plusieurs cibles de manière transparente et tout modèle de mouvement pour les cibles peut y être intégré facilement – il peut même évoluer dans le temps. Le choix d'un modèle de mouvement approprié pour la cible reste cependant, comme ailleurs, un problème non trivial.

Dans une toute autre approche, Riehl et collab. utilisent également un filtrage particulaire pour gérer dynamiquement les cibles et réduire la complexité NP associée aux graphes qu'ils considèrent [147]. Cela aide à aborder le problème de fouille coopérative mais aussi d'étendre la dimension de l'espace de recherche ; il est par exemple possible de considérer non seulement les chemins des robots chercheurs, mais aussi les positions de leurs capteurs lorsque ceux-ci sont orientables.

SYSTÈMES DÉCENTRALISÉS Le lecteur aura peut-être remarqué que la plupart des travaux présentés jusqu'ici en fouille développent des schémas de coordination décentralisés, dans lesquels les robots partagent leurs observations ou leurs plans individuels : il n'y a pas de décisions centralisées ni collectives (avec négociation par exemple). On peut regretter que le partage d'information au sein des systèmes proposés ne soit pas plus développé dans les publications : en effet, maintenir des représentations du monde cohérentes entre les différents robots est un problème de recherche à part entière. Bourgault, Wong et collab. étudient ainsi un problème de fouille aéroportée avec cibles stationnaires ou dérivantes [175] : ils considèrent pour cela des processus bayésiens partiellement indépendants dans un contexte décentralisé. L'ensemble des robots sont considérés comme étant connectés les uns aux autres, mais cette hypothèse semble raisonnable pour une flotte d'AAVs dont les liaisons pair à pair ne sont pas bloquées par des obstacles. Plus récemment, Cole et collab. ont exploités des mécanismes de fusion décentralisée de données (*Decentralized Data Fusion*, ou DDF*) dans un contexte similaire [43] – fouille aéroportée et décentralisée. Les auteurs ont cette fois éprouvés leurs algorithmes sur le terrain à plusieurs reprises.

Une DDF est similaire à un filtre de Kalman linéarisé et décentralisé [66].

D'une manière générale, le choix des approches décentralisées est motivé par le gain de robustesse et par la complexité moindre, permettant un plus grand nombre de protagonistes : la complexité est alors généralement linéaire avec le nombre de robots au lieu d'être exponentielle. Cette dernière

propriété s'acquiert cependant au détriment de l'optimalité, celle-ci n'étant alors généralement plus garantie (ou seulement localement). Nous trouvons malgré tout regrettable que la plupart des travaux fassent l'hypothèse d'une pleine connectivité de communication entre les robots, que l'approche soit centralisée ou non. À part dans certains cas comme une équipe constituée de robots aériens seulement ne rencontrant obstacles ni perturbation dans les communications, cela reste peu réaliste. Malheureusement, la validité de cette hypothèse impacte directement la robustesse et l'efficacité des algorithmes. Ce problème est rarement mentionné, et encore plus rarement discuté.

Certains auteurs adoptent néanmoins une autre approche : afin de gérer ce problème récurrent et omniprésent des communications, ils placent ces dernières au cœur de leurs stratégies. Ainsi, Hollinger et collab. ont élaboré une stratégie de coopération implicite avec une connectivité périodique entre les robots [72]. La planification est séquentielle (un robot après l'autre) et linéaire en la taille de l'équipe. Chaque robot planifie son propre plan en fonction des plans déjà existants pour ses co-équipiers, comme illustré figure 8 dans le cas de deux robots. Les auteurs soulignent que la relaxation des contraintes de communications permet des stratégies plus variées et plus efficaces tout en étant plus réalistes dans la prise en compte de la réalité (les communications sont explicitement modélisées). Cela se fait néanmoins au prix de la robustesse aux échecs : entre deux périodes de communications, chaque robot se retrouve seul face aux aléas pouvant survenir. Les auteurs fournissent une comparaison de cette coordination implicite avec un système à coordination explicite basé sur un mécanisme de marché : ce dernier donne de meilleures performances, mais pour un coût prohibitif en calculs et communications.

Hollinger et Singh ont également montré que les problèmes de fouille multirobots ne peuvent être approximés lorsqu'ils sont sujet à des contraintes de communication [70]*. Ce résultat est un argument particulièrement pertinent en faveur des approches de planification décentralisées et séquentielles.

[When subject to communications constraints], "no polynomial-time algorithm can yield a multiplicative performance guarantee unless P = NP". – Hollinger et Singh [70]

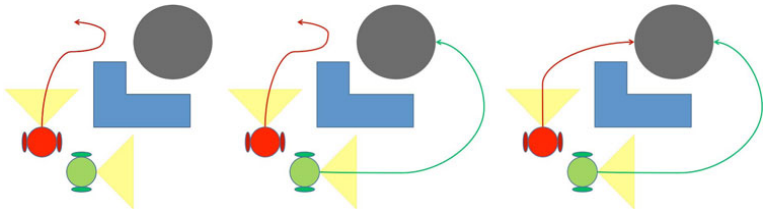


FIGURE 8 – Stratégie de fouille avec une connectivité périodique : *"The robots (green and red) must move around the obstacle (blue L-shape) to observe the area of high information gain (gray circle). They start in line-of-sight contact, and they must regain line-of-sight past the obstacle. The red robot first plans a path that remains connected to the green robot's initial position (left). Then, the green robot plans a path that regains connectivity with the red robot past the obstacle (middle). Finally, the red robot replans to regain connectivity with the green robot's new path (right)."* – tiré de [72]

OPTIMISATION La plupart des chercheurs présentent le problème de fouille comme un problème de planification, mais il est possible de le formuler comme un problème d'optimisation (voir aussi la section 4.1.5 à ce sujet). La distribution de probabilité de présence des cibles permet en effet de quantifier aisément l'espace de recherche des solutions, étape nécessaire aux algorithmes d'optimisation.

Cette formulation particulière du problème de fouille est d'ailleurs une des plus anciennes formulations du problème, initialement guidée par l'intérêt des gardes côtes États-Uniens et l'US Navy pour les problèmes de recherche de cibles. Nous renvoyons le lecteur intéressé au livre de Stone [163] dont la première édition en 1975 mentionne ces premiers travaux en *"optimal allocation of effort to detect a target"*. La seconde édition du livre, publiée en 2007, souligne tous les résultats majeurs trouvés depuis, notamment à propos des cibles mobiles, antagonistes ou non*. Les nouveaux résultats concernant les cibles stationnaires ont été bien moins significatifs.

[Since 1975], "on the theoretical side, there has been significant progress made in solving the problem of optimal search for a moving target." – Stone [163], C-Appendix

Parmi les approches sur la recherche de cibles mobiles, on peut citer Ohsumi modélisant la cible comme une particule markovienne stochastique [122], et El-Rayes utilisant un modèle de mouvement brownien pour trouver une stratégie optimale de recherche [54]. Tous deux considèrent des scénarios monorobots / monocibles, le premier sur un cercle et le second sur une ligne Ces travaux donnent des garanties théoriques sur les stratégies proposées, mais manquent de validations expérimentales. L'hypothèse selon laquelle les fonctions considérées sont continues est d'ailleurs une limite majeure de ces travaux, ces fonctions étant rarement continues en pratique, notamment pour la visibilité. C'est pourquoi Israel et collab. proposent un algorithme presque optimal permettant explicitement de gérer ces discontinuités [75].

Face à la complexité des problèmes de fouille, certains auteurs cherchent à proposer des solutions presque optimales. Sarmiento et collab. présentent par exemple une approche en deux étapes [157] : dans un premier temps, ils considèrent un problème de fouille simplifié, approximée à l'aide d'un ensemble de courbes critiques aux formes précalculées. Dans un second temps ils raffinent cette solution qualitative et optimisent localement le chemin de fouille.

Gan et collab., pour leur part, proposent un algorithme basé sur une descente de gradient décentralisée afin de coordonner une équipe d'AAVs prévenant explicitement les collisions entre eux [60]. Comme souvent avec les AAVs et les approches basées optimisation, l'environnement est considéré sans obstacle pour les robots ; les cibles peuvent en avoir. Le problème de fouille peut également être formulé comme un problème d'optimisation linéaires en nombres entiers (*Mixed-Linear Optimisation Problem*, ou MILP) [59]. Ce type de problème est bien connu et très étudié : une fois formulé, il existe de nombreux solveurs disponibles permettant de trouver des solutions.

Because of the computer developments "the trend in search theory is toward algorithms for computers and away from the theorem-proof style of presentation given in [163]" – Stone, ibid.

Enfin, comme Stone l'a souligné dans son étude*, les récentes avancées en puissance de calcul ont amené de nouvelles approches de résolution des

problèmes d'optimisation. Cette tendance s'illustre par les récents développement des algorithmes « en essaim » appliqués aux problèmes multirobots [140, 67, 16]. Les résultats de ces algorithmes sont intéressants en simulation, mais leur application sur le terrain n'est pas évidente car ils reposent souvent sur de fortes hypothèses concernant les communications (comme une émulation du système de phéromones pour les insectes). Nous discutons plus en détails de cet aspect dans la section 4.2.1.

DÉCISION MULTICRITÈRE L'utilisation des techniques d'optimisation en robotique a pour principal avantage que les solutions obtenues s'appuient sur des fondements théoriques solides. Malgré tout, Amigoni critique l'usage trop répandu de fonctions d'utilité *ad hoc* dans le calcul des plans des agents [11]. En effet, les utilités *ad hoc* proposées semblent donner des résultats satisfaisants et exhiber les comportements attendus, mais elles manquent de justifications théoriques et s'adaptent mal aux changements dans les préférences de comportements. Elles sont en effet fixées « à la main », au jugé ou de manière empirique. C'est pourquoi Amingoni et collab. proposent comme alternative d'introduire des pratiques issues de la décision multicritère. Cela permet, entre autres, de mieux gérer les fronts de Pareto*, et de mieux agréger les différents critères d'évaluation, par exemple au travers d'intégrales de Choquet. Selon les auteurs, les techniques de décision multicritère exploitent mieux les informations disponibles que les utilités *ad hoc* usuelles, sous réserve bien sûr que ces informations soient précises et pertinentes. Les résultats expérimentaux présentés par Amigoni semblent effectivement avaliser cette approche en montrant des gains de performances significatifs [13].

En décision multicritère, un optimum de Pareto est une solution dont on ne peut améliorer un critère sans en détériorer un autre. L'ensemble de ces optimums est appelé front de Pareto.

RÉSUMÉ Les méthodes probabilistes de recherche de cibles mobiles sont relativement récentes : elles s'appuient sur des travaux plus anciens en couverture de zones et sur la recherche de cibles statiques. Il existe de nombreux travaux prometteurs sur le sujet, ce type d'approche étant en grand partie soutenu par les capacités de calculs actuelles. À cause de la complexité du problème de recherche et détection de cible, il est difficile d'obtenir des garanties au pire cas. Les algorithmes proposés s'appuient alors sur les travaux disponibles en mathématiques appliquées (sur les probabilités, la théorie des jeux, la décision multicritère, les techniques d'optimisation, *etc.*). En outre, les spécificités des problèmes de fouille font qu'ils sont plus proches des conditions rencontrées sur le terrain que les autres classes de problèmes de détection de cibles. Ils sont particulièrement adaptés aux manques de moyens (robots, temps) et au besoin de définir des priorités entre les tâches, par exemple dans les scénarios de *Search and Rescue*. Les approches probabilistes permettent également de faciliter l'intégration des algorithmes à bord des robots car elles prennent naturellement en compte les incertitudes omniprésentes dans le monde réel [171].

2.4 PATROUILLE

To patrol : "to keep watch over (an area) by regularly walking or travelling around it". – Oxford dictionnary

Les problèmes de fouille consistent à rechercher une ou plusieurs cibles dans une zone donnée tout en donnant certaines garanties de les trouver effectivement. Les problèmes de patrouille sont similaires mais ajoutent un aspect cyclique voire perpétuel aux problèmes de fouille : la zone est parcourue plusieurs fois en boucle. La notion de régularité y est centrale*. Bien que les problèmes de patrouille soient liés aux problèmes de couverture de zones, de capture et de fouille, il est important de noter que parcourir optimalement une zone ne permet pas de définir un chemin de patrouille optimal, comme illustré figure 9. Dans cette section, nous nous intéressons essentiellement à la patrouille de zones – en deux dimensions – ou de points de passages plutôt qu'à la patrouille de périmètres – à une seule dimension. La patrouille de zones est par ailleurs souvent ramenée à un problème de patrouille de points de passage, proche du TSP. Tout comme en fouille probabiliste, le but est de détecter ou « voir » les cibles : nous n'aborderons donc pas ce que les robots devraient faire une fois les cibles détectées.

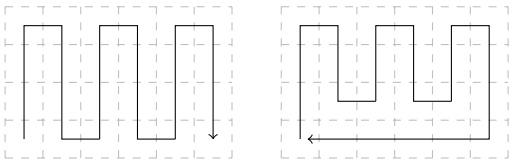


FIGURE 9 – Le parcours optimal d'une zone n'est pas nécessairement à l'origine d'un chemin de patrouille optimal : en effet, le premier ne prend pas en compte l'aspect cyclique du problème de patrouille, qui souvent incite à trouver des chemins formant une boucle correspondant aux cycles de patrouille – librement inspiré de [63].

Les problèmes de patrouille n'ont fait leur apparition que dans la dernière décennie mais ont néanmoins été intensivement étudiés. Peu après les premiers travaux sur le sujet, Almeida et collab. ont proposé une étude complète, définissant les bases des futures travaux dans le domaine [9]. Ils proposent un critère d'évaluation à minimiser appelé « oisiveté » (*idleness*) qui correspond au temps écoulé entre deux visites consécutives au même endroit (point ou région) de la zone à patrouiller. Ce critère renvoie directement à la notion de régularité mentionnée plus haut. Bien que d'autres métriques aient été proposées depuis, telles que les performances face à différents types d'intrusions [155], l'oisiveté et ses variations (en moyenne, au pire cas, normalisée, *etc.*) restent le principal critère d'évaluation des performances dans le domaine.

Il a par ailleurs été observé expérimentalement [9, 137] et prouvé formellement [55, 130] que la solution optimale par rapport au critère d'oisiveté était obtenue en résolvant un TSP sur l'ensemble de la zone afin de trouver le cycle hamiltonien correspondant. En multirobot, il suffit alors de

distribuer les robots régulièrement le long du chemin obtenu. Résoudre le problème de patrouille est donc NP-difficile dans le cas général, bien que la solution optimale puisse être trouvée polynomialement pour certaines topologies spécifiques [130]. Ce degré de complexité, parmi d'autres considérations, encourage la recherche d'autres stratégies.

Almeida a également montré que les stratégies de type « marcheur aléatoire » ont de piètres performances, et qu'il est donc nécessaire de proposer des solutions plus élaborées [9]. Dans une étude plus récente, Portugal et collab. ont identifié plusieurs types d'approches [137]. Ci-dessous, nous suivons, résumons et complétons brièvement cette étude.

CYCLE HAMILTONIEN Les approches cycliques basées sur le TSP cherchent à calculer le cycle hamiltonien d'un graphe utilisé comme représentation topologique de la zone à patrouiller. Les robots sont ensuite positionnés le long du cycle, espacés régulièrement entre eux. Ils suivent tous le même chemin, les uns à la suite des autres, en canon. Pour cette raison, la solution est précalculée hors-ligne : d'une part à cause de la difficulté intrinsèque d'un TSP* d'autre part parce que les robots ne se croisent jamais (sauf incidents ou aléas) et sont donc obligés de se mettre d'accord *a priori* sur la route à parcourir. Le travail d'Elmaliach est une très bonne illustration de cette approche [55].

Le TSP est un problème NP-difficile, mais de nos jours certains programmes peuvent résoudre en quelques minutes de larges instances, de quelques centaines voire milliers de nœuds – la topologie de l'instance influençant beaucoup la complexité de sa résolution.

Bien qu'optimal par rapport à l'oisiveté, ce type de solutions a différents revers. Tout d'abord, il manque de souplesse : il n'est pas capable de faire face aux environnements dynamiques ni de prendre en compte des contraintes telles que des communications périodiques ou la charge des batteries. Il n'est pas non plus adéquat lorsque l'équipe de robots est hétérogène*. De plus, c'est un algorithme centralisé, et calculer un cycle hamiltonien peut s'avérer complexe pour certaines topologies ou tailles d'environnement, même hors-ligne. Il existe des algorithmes d'approximation en temps polynomiaux avec garanties de performances [130], mais ces algorithmes sont en pratique dépassés par d'autres approches (bien que sans garanties) [9, 138]. L'état de l'art sur le TSP contient des algorithmes très efficaces, mais quand le graphe de patrouille n'est pas cyclique, résoudre un TSP ne donne souvent pas une solution adéquate [138].

"The cyclic strategy theorically could reach the minimum maximal refresh time [...] but only for teams of homogeneous robots and assuming no communications constraints."
– Acevedo et collab. [1]

Alamdari et collab. proposent d'introduire des niveaux de priorités entre les différents sommets du graphe de déplacement : ils désignent cette variante comme un *min-max latency walk problem* [4] dans lequel ils essaient de minimiser l'oisiveté maximale, celle-ci étant pondérée par les niveaux de priorité. Cette formulation originale apparaît comme un peu plus complexe que la formulation habituelle. La résolution par TSP seul est d'ailleurs sous-performante, car elle n'est pas capable de prendre en compte ces niveaux de priorités. Ils proposent pour cette variante du problème un nouvel algorithme qui casse la complexité en permettant d'aborder chaque niveau de priorité de manière indépendante. À niveau de priorité égal, la solution d'un TSP reste optimale. Ainsi, de manière paradoxale, l'ajout de nouvelles

contraintes (les niveaux de priorité) permet de réduire la complexité résultante du problème.

Outre leur caractère optimal, les solutions du problème de patrouille par cycle hamiltonien ont pour propriété intéressante d'être indépendantes par rapport au nombre de robots. Néanmoins, le lecteur attentif aura sans doute remarqué que nous n'avons pas mentionné les communications pour les travaux sur la patrouille présentés jusqu'ici : c'est parce qu'elles ne sont pas prises en compte dans les algorithmes mis en avant. Les robots ne coopèrent pas vraiment dans ce contexte et agissent indépendamment les uns des autres, ce qui peut être problématique en contexte opérationnel.

De plus les solutions par cycle hamiltonien ne sont pas adaptées dans un contexte de sécurité avec des cibles antagonistes et intelligentes : les solutions sont en effet déterministes, donc pleinement prédictibles. Il est alors aisé pour des intrus attentifs de passer au travers de tels schémas de patrouille. Ce dernier point souligne un besoin d'imprédictibilité, qui passe par des solutions stochastiques et sous-optimales. C'est le point de départ d'autres approches, et cela justifie également l'usage de métriques alternatives et complémentaires de l'oisiveté [155].

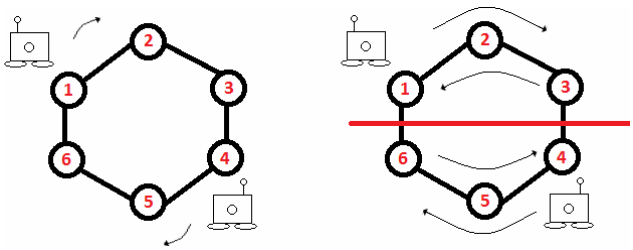


FIGURE 10 – Exemples de stratégies par cycle hamiltonien (à gauche) et par partition (à droite) – tiré de [137].

PARTITIONNEMENT Les cycles hamiltoniens sont des conditions optimales sous certaines conditions et permettent d'aborder le problème de patrouille dans son ensemble. Leur alternative la plus courante consiste à partitionner la zone de patrouille en plusieurs sous-régions, puis à assigner un agent à chacune de ces sous-régions (voir la figure 10). Il existe des variations dans la façon de partitionner la zone de patrouille, et dans la façon de patrouiller chacune des sous-régions. Généralement, la zone de patrouille est modélisée par un graphe, l'approche reposant alors sur le riche état de l'art en partitionnement de graphe. Il est également possible d'utiliser des stratégies par partitionnement de chemins, mais ces approches sont moins efficaces que les approches par partitionnement de régions [1].

Ce type d'approche est notamment illustré par les travaux de Portugal, qui a beaucoup travaillé sur le sujet. Lui et ses collaborateurs proposent l'algorithme MSP (pour *Multilevel Subgraph Patrolling*) qui divise le graphe en autant de sous-graphes que le nombre de robots, et définit pour chacun d'eux un chemin de patrouille localement optimal [136]. Le graphe de départ est construit à partir d'une grille représentant l'environnement. Les résultats

expérimentaux montrent de bonnes performances, proches de l'optimal obtenu par un cycle hamiltonien. Néanmoins, il faut noter que cette approche reste déterministe et que les robots agissent toujours indépendamment les uns des autres – hormis pour la répartition initiale.

L'approche de Fazli décrite dans [57]* repose sur les traditionnels problèmes de couverture de zones et de gardiens de musée : ces derniers permettent de définir un ensemble de points de contrôle sous-tendant une représentation topologique par graphe de la zone de patrouille. La carte topologique est alors construite par triangulation (*Constrained Delaunay Triangulation*) au lieu de la traditionnelle grille. Cela permet de mieux prendre en compte la portée limitée des capteurs des agents patrouilleurs. Les chemins de patrouille sont alors construits à partir d'arbres couvrants. L'algorithme est complet et possède des garanties sur sa complexité.

L'article de Fazli et collab. [57] ne mentionne pas explicitement le terme de patrouille, mais rentre néanmoins parfaitement dans la définition que nous en donnons.

Lorsque les cibles recherchées sont explicitement non antagonistes voire immobiles, le problème de patrouille est également désigné sous le terme de surveillance permanente (ou *persistent monitoring*). Soltero propose de résoudre ce problème par une approche basée sur une partition, en optimisant une fonction de coût donnée [162]. Le résultat est une loi de commande pour les robots qui prend en compte les différents niveaux d'intérêt dans la zone à patrouiller. Il est intéressant de noter qu'il n'y a ici aucune structure de graphe : les résultats sont valables pour des modèles continus. En outre, les approches par partitionnement s'adaptent assez bien aux systèmes décentralisés, comme illustré par les travaux d'Acevedo [1].

SYSTÈMES MULTI-AGENTS Un autre type d'approches repose lui sur des mécanismes classiques en systèmes multi-agents. Les algorithmes d'enchères forment en effet une solution efficace aux problèmes de patrouille, bien qu'ils restent soumis aux critiques habituelles, notamment celles liées aux problèmes de communication. Ce type d'approches a généralement des propriétés intéressantes telles que la complexité face au nombre de robots, la robustesse, l'aspect décentralisé et la flexibilité. Nous renvoyons le lecteur désirant approfondir le sujet à l'étude de Portugal [137], ainsi qu'aux travaux de thèse de Cyril Poulet [139], ces derniers présentant une emphase sur les différents mécanismes de coopération entre agents.

Portugal et ses collaborateurs ont récemment proposé une approche multi-agent distribuée au problème de patrouille [138]. Ils mettent en avant la nécessité de minimiser les interférences entre agents afin de minimiser l'impact du nombre d'agents sur la complexité. Ils utilisent différents modèles bayésiens et comparent leur algorithme aux travaux antérieurs. L'approche semble robuste aux échecs tout en montrant des performances intéressantes. Les algorithmes ont notamment été implémentés au sein de vrais robots et éprouvés en environnement d'intérieur. On peut regretter le manque d'analyses statistiques des résultats, mais deux points sont intéressants à noter : d'une part, leur implémentation de l'approche par cycle hamiltonien n'est pas celle qui montre les meilleurs performances, ce qui n'est pas surprenant dans la mesure où les environnements testés n'ont pas

Lorsqu'un graphe n'est pas cyclique, il n'existe pas de cycle Hamiltonien au sens strict : il est nécessaire de repasser par certains sommets afin de couvrir tout le graphe, ce qui est cause de non-optimalité.

une topologie cyclique*. D'autre part, leur implémentation de l'approche par partition s'avère être la meilleure, en particulier lorsque le nombre de robots augmente au dessus de dix. Ceci souligne l'importance des considérations topologiques et de la composition des équipes de patrouille dans le choix des approches.

STRATÉGIES À BASE DE MARQUEURS Certains des premiers travaux [9] et des approches par essaim plus récentes [40] utilisent des marqueurs d'environnement pour échanger des informations entre agents et coordonner ces derniers. Les approches en essaim sont notamment inspirées par le fonctionnement des phéromones de fourmis : elles utilisent pour cela une représentation virtuelle de l'environnement, habituellement sous forme de grille dans chaque case de laquelle les agents peuvent sentir et déposer des phéromones. Cela permet aux agents de se guider localement sans nécessiter de grandes capacités cognitives : l'intelligence du système réside dans sa conception : les interactions entre les agents et l'environnement, et la propagation de ces mêmes informations. Ces techniques ne sont pas sujettes aux phénomènes d'explosion combinatoire (avec le nombre d'agents) et prennent en compte dynamiquement tout changement de contexte. Il est intéressant de noter que les solutions résultantes semblent converger invariablement vers un cycle hamiltonien ou une combinaison de cycles hamiltoniens [63].

Glad et collab. ont récemment étudié les principaux problèmes soulevés par les approches en essaim et leur intégration au sein de vrais robots [64]. Cela inclut les notions de synchronicité et d'asynchronicité, les collisions entre agents, la nécessité de pouvoir communiquer localement, la granularité de l'espace et du temps, les interférences, le non-déterminisme, *etc.* Les auteurs montrent que les stratégies à base de marqueurs sont viables mais qu'elles soulèvent de nombreux problèmes sous-jacents durant le processus d'intégration.

APPRENTISSAGE Une autre approche consiste à résoudre le problème de patrouille à l'aide de techniques d'apprentissage, et en particulier d'apprentissage par renforcement appliqué à des processus markoviens – cette méthode est souvent désigné par le terme de *Q-learning*. L'idée est d'apprendre un ensemble de stratégies locales que les agents pourront ensuite réutiliser à loisir tout en s'adaptant aux autres agents, aux cibles et au dynamisme de l'environnement.

Ruan et collab. ont pour leur part adapté les techniques d'apprentissage classiques pour rendre les comportements résultants moins prédictibles [153]. Ils partitionnent la zone à patrouiller et calculent pour chaque sous-région un ensemble de chemins à l'aide d'un apprentissage par renforcement classique (*Q-learning* et processus markoviens). Les agents peuvent ensuite choisir à chaque cycle parmi ces différentes routes précalculées ; afin d'éviter l'aspect déterministe de l'apprentissage, le choix des actions à effectuer – les chemins à suivre – utilise un processus de sélection appelé *softmax* basé

sur une distribution de probabilité des actions possibles, cette distribution étant issue du processus d'apprentissage.

THÉORIE DES JEUX Dans un contexte antagoniste, il est assez naturel de considérer les problèmes de patrouille sous l'angle de la théorie des jeux. Cette dernière est particulièrement adéquate si l'on veut également considérer le comportement des cibles.

Amigoni et collab. se placent dans ce cadre, par opposition aux stratégies stochastiques qui cherchent à être imprédictibles [12]. Selon eux, modéliser le comportement antagoniste des cibles permet d'élaborer de meilleures stratégies en réponse, bien que ces dernières soient sensibles à la qualité de modélisation et aux hypothèses de départ. Dans un contexte monorobot, les auteur modélisent les protagonistes par un processus markovien de premier ordre. La discrétisation du temps et de l'espace est donc cruciale, et il est admis que tout intrus a besoin d'un certain laps de temps pour pouvoir pénétrer dans la zone et accomplir son objectif. Comme c'est l'intrus qui décide quand lancer son attaque, Amigoni et ses collaborateurs proposent de rechercher un équilibre meneur-suiveur (*leader-follower*), c'est-à-dire une stratégie de patrouille qui minimise l'impact d'un intrus réagissant au déplacement de la patrouille. La théorie des jeux fournit de nombreux résultats et propriétés pour ce type d'équilibre, ce qui permet d'élaborer plus facilement une stratégie adéquate. Si les fondements mathématiques d'une telle approche sont solides, on peut néanmoins déplorer le manque de réalisme des hypothèses de travail et la lourde complexité par rapport à la taille de l'environnement.

Il nous semble par ailleurs pertinent de mentionner les travaux de Pita, bien qu'ils soient légèrement en dehors du cadre des applications robotiques et des systèmes autonomes. Le système imaginé par Pita et collab. a été utilisé pendant plusieurs mois comme système d'aide à la décision pour définir les schémas de patrouille de l'aéroport de Los Angeles [135]. Comme souligné par les auteurs, les deux principales difficultés rencontrées par les patrouilles en contexte antagoniste sont d'une part le besoin de patrouiller – ce qui sous-entend un manque de ressource car sinon il serait plus simple et efficace de couvrir toute la zone en permanence, et le besoin d'imprédictibilité, d'aléatoire, car dans ce contexte tout déterminisme est une faille. Le système considère un jeu de Stackelberg* bayésien pour modéliser le problème et élaborer les schémas de patrouille, ou plus exactement les emplois du temps des agents. Le formalisme bayésien est utile pour rendre compte des incertitudes sur les adversaires, tandis que la théorie des jeux de Stackelberg permet d'obtenir des stratégies aléatoires « optimales ». Ce cadre de travail permet de prendre en compte différents niveaux de priorité dans les zones à couvrir et s'accommode bien des processus à « initiatives mixtes », c'est-à-dire des interactions de l'utilisateur avec les systèmes de décisions – cela va des petits ajustements à la réécriture ponctuelle de l'emploi du temps défini automatiquement. Cette dernière propriété est importante dans un contexte opérationnel car elle permet de prendre en compte

Les jeux de Stackelberg sont surtout utilisés en économie pour étudier les contextes de duopôles asymétriques, avec la notion de société « pilote » et de société « satellite ». Ce sont des situations bien connues et très étudiées.

À propos de la difficulté d'être « aléatoire » pour les êtres humains, lire par exemple la synthèse de Brugger [34].

des contraintes externes inconnues du système. Il est largement reconnu* que les humains sont mauvais dans l'écriture de stratégies aléatoires : c'est le principal argument en faveur de l'utilisation d'un système automatisé qui permet de donner certaines garanties à la stratégie finale obtenue. Le système en usage à l'aéroport permettait de définir les points de contrôle « aléatoires » tout comme la répartition des ressources spécifiques (comme les maîtres-chiens). Sous sa forme actuelle, il n'est pas adapté aux systèmes robotiques (pas de prise en compte des chemins, *etc.*) mais il permet de résoudre des problèmes NP-complets – par nature difficiles pour l'être humain – et a été utilisé avec succès en contexte opérationnel pendant des mois, ce qui est une preuve indéniable de son intérêt.

PATROUILLE DE PÉRIMÈTRES Les approches de patrouille évoquées jusqu'ici s'intéressent principalement à des zones en deux dimensions. Il peut être intéressant de se pencher sur des espaces de dimension plus élevée – en trois dimensions ou avec des capteurs orientables par exemple – mais également de dimension plus réduite. En effet, la patrouille de périmètre – espace à une dimension – est un problème classique qui, sous certaines hypothèses, présente un intérêt réel : parfois il n'est pas nécessaire ou du moins peu utile de patrouiller l'intérieur de la zone à protéger. Il arrive également que l'objectif à garder soit une frontière ; on parle alors de périmètre sécurisé. Agmon et collab. s'intéressent à ce problème dans un contexte multirobot et antagoniste [2], le but étant de surveiller le périmètre d'une zone dont il faut prévenir toute intrusion. Ils proposent une solution non déterministe dont l'intérêt a déjà été souligné*. Le périmètre 1D est divisé en segments, l'objectif étant de minimiser la probabilité de pénétration de chacun des segments face à un attaquant intelligent et « observateur », c'est-à-dire connaissant les mouvements des agents patrouilleurs. Les auteurs proposent plusieurs stratégies et étudient leurs propriétés et garanties mathématiques, mais il n'y a pas de validation expérimentale. On notera que la dimension réduite de l'espace de recherche aide à trouver des solutions « optimales » plus facilement que dans le cas à deux dimensions.

"In adversarial settings the frequency criteria becomes less relevant [than non-determinism]". – Agmon et collab. [2]

RÉSUMÉ Les problèmes de patrouille ne sont étudiés que depuis une décennie environ, mais il existe déjà un très riche état de l'art dans le domaine, avec de nombreuses approches et plusieurs études et benchmarks. La vivacité de la recherche dans ce nouveau domaine s'explique par ses liens avec les autres problèmes de détection déjà évoqués dans notre état de l'art et largement étudiés par le passé, mais aussi par le cadre théorique très riche formé par la théorie des graphes et la théorie des jeux. Les graphes sont le modèle d'environnement dominant. Comme déjà soulignés face à d'autres problèmes NP-difficile, la recherche d'approches sous-optimales est importante ; outre le gain de robustesse et de flexibilité, elles apportent aussi un côté imprédictible valorisé par un contexte antagoniste. Les travaux les plus récents mettent en avant ces considérations et étudient principalement des algorithmes locaux tout en s'intéressant aux garanties d'optimalité. On

peut malgré tout regretter que les nombreux résultats théoriques ne soient pas appuyés par des validations expérimentales plus systématiques.

2.5 CHASSE

Les problèmes de chasse sont des problèmes de détection de cibles ne fournissant aucune garantie de détecter la cible. Cette absence de garanties vient généralement d'un manque de moyen plus fort encore que dans les problèmes de fouille ; il s'agit le plus souvent d'un manque d'informations sur l'environnement et les cibles recherchées.

ABSENCE DE CARTES L'absence de cartes peut venir d'un manque d'informations initiales mais aussi d'un choix de conception délibéré. Travailler sans carte – hormis localement pour les déplacements à partir des perceptions courantes – permet en effet d'élaborer des stratégies plutôt robustes aux aléas, incertitudes et changements dynamiques, même si c'est généralement au prix de l'efficacité. C'est une sorte de compromis entre les performances et la robustesse du système, les hypothèses de départ et les informations disponibles servant de curseurs entre les deux bords. Ce choix de conception peut également se justifier par la piètre qualité des capteurs embarqués auxquels il est alors difficile de se fier. Ces problèmes de détection de cibles sans carte globale sont proches des stratégies d'exploration, en particulier lorsque les cibles sont stationnaires. Les stratégies d'exploration sortant du cadre de notre étude, elles ne seront pas abordées dans ce chapitre.

Parmi les autres stratégies, nous pouvons citer les travaux de Cao, qui utilise un système de commande distribuée basé sur des interactions locales appelé *Local Interaction with Local Coordinate Systems* (LILCS) [36]. Ce système permet de résoudre des problèmes de chasse en environnements inconnus. Les robots sont localement coordonnés pour rechercher, suivre et attraper des cibles, la recherche étant globalement aléatoire. Cette stratégie s'avère réactive et robuste aux dérives tout comme aux échecs de communication, bien qu'elle soit largement sous-optimale.

Annas et Xiao présentent eux aussi une stratégie permettant de chercher et suivre une cible en environnement 2D, sans aucun modèle de cible [14]. On notera que la partie de leurs travaux concernant le suivi de cibles se rapporte aux problèmes de suivi abordés à la section 3.2. Ils considèrent en outre que la cible n'a également qu'une connaissance limitée de son environnement. Leur approche a été testée contre des joueurs humains [15].

Carpin et Kolling abordent de leur côté des missions de nettoyage de graphes contaminés similaires aux problèmes de capture mais sans posséder de carte de l'environnement [91] : il n'y a pas de carte initiale, et aucune carte globale n'est construite. S'il y a une recherche de garanties au pire cas, celles-ci ne peuvent être données. Il n'y a en effet aucune considération globale : l'approche est décentralisée, locale et guidée par les frontières perçues. Les robots chasseurs balayent l'environnement en formant une

ligne, comme lors d'une battue. Ceci se justifie par les champs de vue limités ; les robots se déplacent alors en suivant les murs ou les robots voisins. Ce système garantit de balayer toute la zone à couvrir sous réserve qu'il y ait assez de robots par rapport à la largeur de l'environnement, mais il ne peut garantir la décontamination complète de la zone, toute cible mobile pouvant aisément échapper à ces robots.

À mi-chemin entre les stratégies aléatoires non coordonnées et les stratégies de balayage pleinement coordonnées, Miao propose un système auto-organisé [103]. La plupart de modèles auto-organisés s'inspirent des troupeaux d'animaux, des nuées d'oiseaux ou des bancs de poissons, mais ceux-ci ne semblent pas adaptés à la chasse, les agents restants près les uns des autres. Miao propose au contraire un modèle « anti-troupeaux » imitant le comportement social des prédateurs solitaires comme les tigres ou les araignées. De manière inattendue, ce système décentralisé et auto-organisé montre en simulation des performances similaires aux systèmes pleinement coordonnés dans les stratégies de balayage d'une zone. Il se montre pourtant bien plus robuste ; en particulier, il ne requiert pas de communication entre les agents.

LE MANQUE DE RESSOURCES Rabouin choisit d'aborder le problème de recherche de cibles sans posséder assez de robots pour élaborer une stratégie de capture, mais sans non plus faire de considérations probabilistes [143, 144]. Ce dernier point peut se justifier par la sensibilité des stratégies de fouille aux modèles probabilistes retenus (voir la section 2.3). En s'appuyant sur la théorie de l'information, il propose un ensemble d'algorithmes basés sur la géométrie et la théorie des jeux. L'intérêt principal de son approche est de prendre explicitement en compte le fait que les informations disponibles sont incomplètes et imparfaites, alors que la plupart des travaux font souvent l'hypothèse inverse – ce qui d'ailleurs pose des difficultés d'intégration. La stratégie de recherche est guidée par une heuristique construite à partir d'une version simplifiée du problème. Exploitant d'abord un modèle en grille de l'environnement, ses travaux ont ensuite été étendus à des modèles continus – des espaces euclidiens partiellement observables. Ils prennent en compte les échecs de communication. Ses travaux se rapportent aussi aux problèmes de suivi car ils considèrent que les cibles sortent et rentrent régulièrement dans le champ de visibilité des robots chasseurs, mais les positions initiales des cibles sont inconnues et l'objectif principal est bien de les détecter.

Nous avons évoqué à la section 2.3 combien les problèmes de fouille en environnements 3D étaient complexes, plus encore dans le cas d'environnement encombrés d'obstacles – le problème étant déjà au mieux NP-difficile en deux dimensions. Dornhege et collab. montrent que malgré ce caractère insoluble du problème, il est possible de proposer une solution raisonnablement efficace et calculable avec les machines actuelles [48]. L'idée consiste à parcourir une zone 3D à l'aide d'un robot de manière similaire aux scénarios de *Urban Search And Rescue* (USAR) dans lesquels l'aspect temporel

est crucial et où il n'est pas possible d'attendre la fin du calcul de la solution optimale. Considérant le problème comme trop complexe, les auteurs proposent une approche presque optimale à moindre coût. L'algorithme se déroule en deux étapes : on cherche d'abord à trouver un ensemble minimal de poses appelées *views* recouvrant la zone définie. Chaque *view* est constituée de la position 3D du robot et de l'orientation angulaire de ses capteurs. Une fois cet ensemble trouvé, la résolution d'un TSP permet de calculer un chemin optimal dans cette réduction du problème. Les auteurs comparent leur approche à une planification complète sur le problème de départ, qui donne la solution optimale mais à un coût beaucoup plus élevé : les performances de leur algorithme sont expérimentalement très proches de l'optimum bien que sans garantie, tout en ne demandant qu'une fraction du temps de calcul d'origine. Leur approche semble par ailleurs meilleure qu'une approche gloutonne, du moins au niveau du temps d'exécution ; cette tendance est cependant moins marquée lorsque l'on prend en compte la somme des temps de calcul et d'exécution, car le premier est quasi-nul dans le cas des approches gloutonnes.

RÉSUMÉ Les problèmes de chasse ne sont pas autant abordés que les autres problèmes de recherche de cibles mais répondent néanmoins à des besoins précis, souvent issus de contraintes opérationnelles. Les stratégies de recherche présentées sont rarement minutieuses : elles reposent bien souvent sur des décisions gloutonnes ou aléatoires, mais elles forment une base de travail intéressante lorsque l'on manque cruellement de moyens (robot, qualité de capteurs, temps) ou d'informations (cartes, modèles). Les travaux les plus récents gardent ces restrictions tout en présentant des stratégies plus élaborées permettant de gérer ces limitations. Dans les deux cas, les principaux attraits des approches proposées sont leur robustesse et le faible coût d'intégration, que ce soit sur le plan du temps de travail requis ou du coût des plateformes nécessaires, seule une faible qualité étant requise pour les équipements.

3 PISTAGE DE CIBLES

Il faut avoir beaucoup étudié pour savoir peu.

— Charles de Montesquieu

Dans ce chapitre nous rapportons les contributions relatives aux différentes classes de problèmes définies dans la branche « pistage de cibles » de notre taxonomie (voir figure 2 p.7, à droite). Tout comme dans le chapitre précédent, nous rassemblons des travaux issus de différentes communautés (commande, décision, mathématiques appliqués, systèmes multi-agents, robotique, *etc.*) dont nous mettons en évidence les principaux résultats tout en soulignant leur complémentarité. Nous mettons aussi en avant les limites de certaines approches, bien que cela soit développé plus en détails dans le chapitre 4.

Remarque. Ce chapitre est organisé en sections indépendantes les unes des autres, chacune correspondant à une classe de problèmes de détection de cibles définie dans la taxonomie. Chaque section se termine par un résumé ; ces derniers peuvent permettre une lecture plus rapide de ce chapitre.

3.1 LOCALISATION DES CIBLES

Les problèmes de localisation de cibles impliquent une équipe de capteurs mobiles pistant une ou plusieurs cibles à travers plusieurs observations simultanées de chaque cible. Cette multiplication des points de vue permet d'obtenir des informations partiellement redondantes mais surtout complémentaires sur les cibles, avec pour objectif principal d'améliorer la précision de la position estimée des cibles, ou en d'autres termes leur localisation. Ces problèmes sont aussi désignés dans la littérature sous les expressions *Focus of Attention* (FoA) [74] ou *Cooperative Localization and Target Tracking* (CLATT) [105]. Les premières contributions sur le sujet apparaissent au début des années 2000 [74, 44, 66] et ce domaine d'étude a gagné en attention depuis, essentiellement grâce au développement des systèmes multirobots [176, 179, 107, 168, 156].

CONTEXTE COOPÉRATIF De par la nature même du problème, les mécanismes de coopération sont au cœur de la localisation de cibles dont le but est d'exploiter au mieux la variété de points de vue et de capteurs. Il est en outre avantageux d'utiliser des équipes de robots hétérogènes : outre les différences de capteurs éventuellement complémentaires, la coopération entre AGVs et AAVs au sein d'une même équipe [66, 107, 176] permet d'utiliser toutes les composantes de l'espace et de mieux tirer parti des spécificités de l'environnement, comme illustré par exemple par la figure 11.

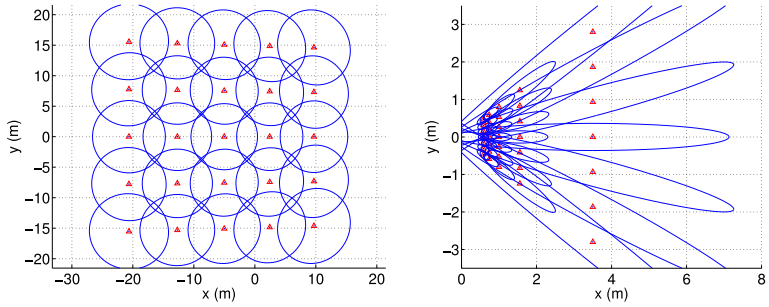


FIGURE 11 – "*Ground feature observation uncertainty from aerial (left) and ground (right) vantage points. The AAV camera looks down 5° off vertical at 50m altitude. The AGV camera is mounted horizontally 0.32m above the ground plane. Comparative feature observation accuracy is illustrated by ground plane confidence ellipses associated with uniformly spaced pixels in the imagery.*" – extrait de [66]. Ceci souligne le potentiel d'une équipe multirobot hétérogène.

Il existe plusieurs approches centralisées déterminant quelles sont les meilleures positions futures à attribuer à chaque capteur [105, 179]. Isler définit une variante du problème appelée *Focus of Attention* (FoA) dont il montre que c'est un problème NP-difficile, complexe à approximer dans le cas général [74]. Pour cette raison, Isler étudie certaines configurations géométriques spécifiques et contraintes (lignes, cercles) pour lesquels il fournit de bons résultats approximés. Son approche est elle aussi centralisée et s'appuie sur un formalisme mathématique fort permettant de donner des preuves et certaines garanties sur la qualité des solutions approximées.

A l'opposé de ces approches centralisées, plusieurs auteurs proposent des méthodes décentralisées [44, 66, 80, 177], dans lesquelles ils traitement localement les aspects de coordination. Pourtant, comme constaté par Moseley, un système pleinement décentralisé n'est souvent utile que lorsqu'il implique un grand nombre de robots – quelques dizaines ou plus. Les approches centralisées des problèmes de localisation restent quant à elles abordables pour de petites équipes de robots, et s'avèrent généralement optimales, même si elles ont aussi d'autres inconvénients, notamment en ce qui concerne les communications. Moseley propose donc un système hybride comme un compromis entre ces deux extrêmes [107].

Les problèmes de localisation de cibles ne se réduisent cependant pas aux seuls mouvements des robots par rapport aux cibles : ils posent aussi des problèmes de fusion de données. Ces dernières viennent en effet de sources multiples et variées, et nécessitent d'être fusionnées pour tirer pleinement partie des synergies multirobots. De nombreuses méthodes ont été proposées pour effectuer cette fusion de données dans le contexte étudié ; nous pouvons citer notamment les *Extended-Kalman Filters* (EKF) [105], la *Decentralized Data Fusion* (DDF)* [120, 66, 107], les *Rao-Blackwellized particles filters* (RBPF) [141], le *Multiple Hypothesis Tracking* (MHT) [168] ou encore les *filters on Gaussian Mixtures Models* (GMM) [156].

PROBLÈMES D'INTÉGRATION Comme mis en avant par Cowley [44], les problèmes de communication et le contexte de base de données distribuées sont cruciaux lorsqu'on aborde des problèmes multirobots sur des plateformes embarquées. C'est d'autant plus vrai dans le domaine de la localisation où il n'est pas possible de trouver de solution de contournement, le partage et la fusion des données étant au cœur même du problème. Ce constat explique certainement pourquoi la plupart des travaux abordant la partie intégration sur plateformes robotisées porte directement sur le calcul distribué et la gestion de bases de données distribuées. Cowley souligne en particulier l'énorme écart entre la taille des données – elle-même multipliée par le nombre de robots – et le débit de communication qui reste invariablement bas ; en fait, ce dernier se réduit même lorsque le nombre de robots impliqués augmente. Ceci soulève l'intérêt d'avoir des bases de données distribuées et de n'échanger que des données résumées ou prétraitées. Il est également possible de réduire dans une certaine mesure le besoin de communiquer pour se coordonner : Xu et collab. introduisent par exemple des méchanismes de coordination implicite basés sur des techniques d'apprentissage [177].

Il est par ailleurs intéressant de noter qu'il y a eu beaucoup d'efforts pour intégrer les algorithmes en localisation de cibles au sein de vrais robots [66, 107, 156, 168, 177, 105], bien que cela soit à relativiser en portant une attention particulière aux conditions expérimentales. En effet, comme déjà mentionné par ailleurs, certaines simulations ou expériences avec de vrais robots ne reflètent que partiellement les algorithmes qui sont présentés et manquent de réalisme quant aux conditions expérimentales [80].

VARIATIONS La plupart des approches de localisation de cibles sont monocibles mais certaines considèrent plusieurs cibles simultanément [80, 74, 168]. Dans de tels cas, le problème est fortement lié à celui d'observation (voir la section 3.3 pour plus de détails). Cela rajoute une dimension au problème : la répartition des cibles entre les différents capteurs.

Une autre variation intéressante consiste à considérer une cible changeante, ou plus précisément qui change de forme : cela peut être un feu de forêt, un fuite de pétrole ou de produits chimiques, un troupeau d'animaux ou encore une foule de personnes. La cible présente alors certaines spécificités que l'on peut prendre en compte pour concevoir des algorithmes plus efficaces. Par exemple Casbeer et Kingston considèrent spécifiquement la surveillance de feux de forêt, de manière distribuée avec une équipe d'AAVs [37], tandis que Clark and Fierro présentent un algorithme de détection de périmètre et de suivi de fuites de substances chimiques [42]. Ces derniers travaux sont d'ailleurs liés à ceux de couverture de périmètres, où les robots « gardent » les frontières de la cible (voir la section 2.1). Les algorithmes de Clark et Fierro sont également liés aux problèmes de chasse (voir la section 2.5), car au début de la mission les robots cherchent aléatoirement ces substances dans la zone considérée. Cette partie reste néanmoins anec-

dotique, mais ce rapprochement illustre lui aussi les liens forts entre les parties gauche et droite de notre taxonomie.

OPTIMISATION Les problèmes de localisation de cibles bénéficient également des formulations sous forme de problèmes d'optimisation : Kamath et collab. les présentent par exemple comme des problèmes de minimisation d'énergie, très étudiés en optimisation [80]. Ils proposent un algorithme de résolution itératif et distribué. Xu pose pour sa part le problème de coopération comme un problème traditionnel d'optimisation non linéaire en nombres entiers, aussi appelé *Mixed Integer Non-Linear Programming* (MINLP) [176]. On ne connait pour ce type de problèmes, bien étudié également, que des solveurs centralisés. Xu propose donc d'approximer le problème d'origine pour le réduire à un problème de programmation non linéaire simplement, pour lequel il existe des méthodes de résolution en temps réel et décentralisées.

RÉSUMÉ Les problèmes de localisation de cibles présentent plusieurs facettes, chacune étant complexe à aborder : la position des capteurs par rapport à la cible, la coopération multirobot, la fusion de données, les problèmes de communication, *etc.* Beaucoup de travaux proposés s'appuient sur des résultats déjà obtenus dans d'autres contextes par différentes communautés (parmi eux : les réseaux de capteurs, la commande, la théorie de l'information, les systèmes distribués). Les approches centralisées montrent de meilleures performances que les approches décentralisées qui sont souvent sous-optimales, mais les premières ne sont pas toujours réalistes dans leurs hypothèses de travail et sont rarement validées expérimentalement. La complexité du problème empêche par ailleurs de le résoudre optimalement en ligne. Il existe cependant plusieurs implémentations fonctionnelles des approches distribuées, ce qui semble avaliser ce choix de conception.

3.2 SUIVI

D'après notre taxonomie, nous définissons les problèmes de suivi comme étant le fait de pister une seule cible avec un seul robot. L'objectif est généralement de maintenir la cible en vue sans que celle-ci ne coopère à cet effet – les déplacements de la cible ne sont d'ailleurs généralement pas connus. Les problèmes de suivi ont été largement étudiés, avec notamment beaucoup de travaux mathématiques basés sur une description très géométrique du problème [41, 6, 119].

TRAVAUX PIONNIERS Les premiers travaux dans le domaine s'y réfèrent comme un problème de poursuite-évasion, ou encore problème du Lion et de l'Homme (*Lion and Man problem*). Pour un état de l'art plus approfondi de ce dernier, le lecteur est renvoyé à l'introduction de [121]. Les travaux pionniers furent également motivés par les scénarios de conflits maritimes.

Il est communément reconnu que se déplacer droit vers la cible est une stratégie naïve, et qu'il est nécessaire pour le poursuivant de prendre en compte les obstacles afin de se déplacer de manière optimale [117, 18]. De manière plus générale, les stratégies basées sur l'asservissement visuel à la cible, comme dans [126], n'ont qu'un succès limité car elles ne prennent pas en compte les particularités de l'environnement. Les algorithmes présentés ci-après et construits sur des considérations géométriques vont au-delà de ces limitations. Ils forment la majeure partie des travaux dans le domaine et sont à l'origine des meilleurs résultats démontrés et observés.

En outre, et à l'opposé des asservissement visuels, une formulation trop abstraite peut conduire à des résultats non applicables en pratique. Ainsi, dès 1962, Eaton et Zadeh proposent d'utiliser des modèles markoviens pour élaborer la meilleure stratégie permettant d'attraper une cible, fixe ou mobile [50]. Malheureusement, leur approche se base sur de nombreux paramètres et fonctions mathématiques non explicités, et qu'il n'est pas facile d'instancier dans le monde réel. Leurs résultats sont en quelque sorte trop « éthérés » pour pouvoir être exploités*.

Ce problème d'abstraction à sens unique ne permettant pas un retour à la réalité est un problème récurrent que nous analysons plus en détails au chapitre 5.

CONTRAINTES DE VISIBILITÉ Comme expliqué précédemment, les problèmes de suivi imposent généralement au poursuivant de maintenir la cible en ligne de mire : il est pour cela nécessaire d'évaluer quelles sont les contraintes de visibilité. Même dans les cas où une visibilité continue n'est pas nécessaire, il reste nécessaire de considérer ces contraintes car perdre la cible de vue peut conduire à ne jamais retrouver sa trace. La visibilité du poursuivant et ses stratégies sont contraintes par les obstacles et par le champ de vision du capteur utilisé, et notamment sa portée.

Ainsi, LaValle essaie de planifier des chemins optimaux maximisant la visibilité sur une cible prédictible, le tout en environnement discret [94]. Murrieta-Cid propose lui aussi de maximiser la visibilité sur la cible, mais de manière réactive et face à des cibles imprédictibles [117]. Il introduit pour cela la notion de plus courte distance de fuite, appelée SDE pour *shortest distance to escape*. Derrière cette notion se cache une idée intéressante : celle que la cible, pour se cacher, doit se rapprocher des obstacles ; plus les obstacles sont proches, et plus le risque de perdre la cible est grand. On retrouve la même idée dans les notions de *vantage time* et de *gap-zone* introduites par Bandyopadhyay et collab. [18]. La *gap-zone* est déterminée par les obstacles autour de la cible ; elle est également exploitée de manière gloutonne et réactive. Cette dernière stratégie se montre bien meilleure que la stratégie naïve déjà évoquée (*"run to the target"*). Implémentée sur un robot, ce dernier est capable de suivre un humain de déplacement dans une foule, c'est-à-dire dans un environnement encombré et dynamique, et ce en temps réel. Cette validation expérimentale est une excellente preuve de la maturité de ce type de solutions.

Plus récemment, Noori et collab. ont ajouté des contraintes de visibilité au problème classique du Lion et de l'Homme [121], en considérant un champ de vision limité pour les deux protagonistes. À vitesses égales, ils

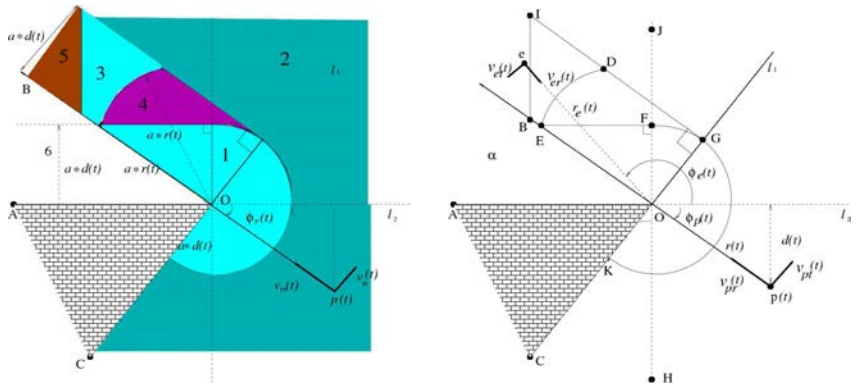
"A single deterministic pursuer with line-of-sight visibility can capture an evader whose speed is equal to the pursuer's in any monotone polygon." – Noori et collab. [121]

ont montré qu'une stratégie déterministe permettait d'attraper la cible dans tout environnement polygonal monotone*, cette dernière propriété étant cruciale pour la validité du résultat (en particulier, le résultat n'est pas valable si le polygone est seulement faiblement monotone).

STRATÉGIES LOCALES OPTIMALES ET COMPLEXITÉ Bhattacharya et collab. ont étudié en détails la configuration locale atomique du problème, composée d'un poursuivant, d'une cible et d'un obstacle polygonal, le tout en environnement continu [28]. Les auteurs découpent géométriquement cette configuration pour y distinguer plusieurs zones, chacune correspondant à une stratégie optimale de suivi comme illustré figure 12. Plus récemment, Karnad et Isler ont proposé des résultats similaires dans le cas d'un obstacle circulaire [82]. Connaissant les positions initiales des protagonistes, il est possible de déterminer qui gagnera le jeu de poursuite associé (la cible essayant d'échapper au poursuivant). Cette approche présente néanmoins un revers : sa complexité augmente drastiquement avec le nombre d'obstacles. De manière plus générale, le problème de suivi est entièrement décidable, mais définir une stratégie globale optimale est NP-complet [116], car il faut résoudre un TSP sous-jacent.

"Although analytical solutions of some simple pursuit-evasion games are known, most interesting instances can only be solved using numerical methods requiring significant off-line computation." – Karaman et Frazzoli [81]

Face à cette complexité, Karaman et Frazzoli proposent un algorithme permettant de définir une stratégie d'évasion à partir d'un échantillonnage incrémental [81]. Leur méthode est donc avant tout numérique plutôt qu'analytique*. De manière intéressante, l'algorithme s'avère être *anytime*, et les auteurs démontrent qu'il est correct et probabilistiquement complet.



(a) Partition définissant différentes stratégies de suivi

(b) Partition géométrique

FIGURE 12 – Partition géométrique de l'environnement basée sur les stratégies de suivi – proposé par Bhattacharya [28]. Cette partition se base sur la géométrie de l'obstacle, la position de la cible et la différence de vitesse entre la cible et le poursuivant : elle met en évidence différentes zones, chacune correspondant à une stratégie optimale différente pour le poursuivant.

CINÉMATIQUE ET ENVIRONNEMENTS COMPLEXES Plus récemment, certains auteurs ont étudié spécifiquement la cinématique des robots poursuivants et les contraintes associées à leurs capacités de mouvement. En effet, des difficultés supplémentaires peuvent apparaître lorsqu'il existe une différence d'agilité entre la cible ou le poursuivant. Ces considérations cinématiques sont fréquentes lorsque le poursuivant est un AAV à voilure fixe [167], même si les modèles restent simples voire simplistes. Ce type de considération est en revanche beaucoup plus rare pour les AGVs, bien que beaucoup d'entre eux soient des plateformes non holonomes. Néanmoins, l'importance et l'impact de ces considérations sont réels ; le lecteur désirant approfondir le sujet pourra se reporter aux travaux de Murrieta-Cid qui propose une stratégie de poursuite pour un robot à essieux poursuivant une cible omnidirectionnelle [118].

Outre la complexité induite par les protagonistes et leurs modèles de visibilité et de déplacements, l'environnement possède lui aussi une complexité sous-jacente. En fait, la plupart des auteurs considèrent un environnement 2D pour lequel les obstacles à la vision sont confondus avec ceux pour le déplacement, et partagés entre la cible et le poursuivant. Pourtant, dans les environnements réels, en 3D, de nombreuses difficultés apparaissent, comme des bâtiments ou des sous-bois qui gênent la visibilité des AAVs mais pas leurs déplacements. De manière plus générale, les obstacles aux mouvements et ceux pour la visibilité sont distincts lorsque l'on aborde des applications réelles et concrètes. Récemment, plusieurs travaux ont fait l'effort de considérer des environnements plus complexes : Bhattacharya a par exemple cherché à réduire des environnements 3D à deux dimensions seulement, par projection [28], mais a aussi effectué plusieurs simulations exploitant pleinement la 3D [22]. Nous avons de notre côté proposé des représentations 2.5D et multiniveaux pour enrichir les modèles de visibilité et de mouvements [148]*. Bien entendu, exploiter des modèles d'environnement de plus grande expressivité a un coût : l'espace de recherche des solutions est de plus grande dimension, ce qui s'avère critique quand les problèmes de départ ont déjà une grande complexité combinatoire.

Nos travaux publiés dans [148] font l'objet du chapitre 9.

Outre le besoin d'utiliser des modèles plus réalistes, il est également intéressant de considérer des modèles beaucoup plus abstraits que les mondes 2D euclidiens. Ainsi, le problème historique du Lion et de l'Homme continue de motiver les mathématiciens. Alexander a récemment étendu certains résultats de poursuite-évasion en espace euclidien aux espaces CAT(k)*, notamment pour $k = 0$ [8, 7]. Ces espaces ne sont probablement pas familiers pour beaucoup de roboticiens, mais il est souvent peu difficile de généraliser les résultats des espaces euclidiens aux espaces d'Hadamard (CAT(0)). L'intérêt selon Alexander est qu'il est parfois plus facile de trouver des solutions directement dans les espaces CAT(k) car ces derniers sont moins contraints. Ainsi, certains algorithmes se retrouvent polynomiaux au lieu d'être NP-difficile. Cette promesse est attirante, mais les auteurs ne fournissent aucun résultat expérimental concret venant appuyer leurs propos.

Un espace CAT(k) est une classe spécifique d'espace métrique.

MODÈLES DE CIBLES Comme expliqué précédemment, la plupart des travaux en suivi de cibles présupposent que la cible et le poursuivant ont des capacités de déplacement similaires. Le comportement de la cible est par ailleurs souvent considéré comme imprédictible, avec une résolution au pire cas, ce qui n'empêche pas de planifier une stratégie pour un horizon temporel fini, le comportement de la cible étant alors « indépendant » de celui du poursuivant [148]. Il est également fréquemment présupposé que la cible s'oriente naturellement par rapport aux obstacles et garde un cap globalement constant [18]. Le comportement de la cible peut également être modélisé par des processus markoviens (POMDP) [21], comme mentionné ailleurs dans le chapitre 2, bien que le surcoût en calculs contraigne alors fortement les tailles de l'environnement et de l'horizon temporel considérés.

VARIATIONS Dans la formulation traditionnelle du problème de suivi, le poursuivant essaie « d'attraper » la cible qui, elle, tente de lui échapper. « Attraper » se traduit ici par le fait d'être aussi proche que possible de la cible, une distance nulle signifiant que la cible est attrapée. Il existe néanmoins plusieurs variantes : l'une d'elle consiste notamment à maintenir une distance fixe avec la cible [111] ou encore à garder ses distances sans être trop loin. Il existe en effet de nombreuses situations dans lesquelles on préfère garder ses distances, par exemple pour des raisons de sécurité ou pour avoir une meilleure vue d'ensemble sur la cible.

En théorie des jeux, l'équilibre de Nash est un concept de solution dans lequel l'ensemble des choix faits par plusieurs joueurs, connaissant leurs stratégies réciproques, est devenu stable du fait qu'aucun ne peut modifier seul sa stratégie sans affaiblir sa position personnelle.

Inspiré par le problème du déménageur de piano [158], Murrieta-Cid et collab. définissent quelles sont les conditions suffisantes pour échapper à une poursuivant doté d'une distance de visibilité limitée, ce qui leur permet d'élaborer une stratégie d'évasion [113]. De manière similaire, Bhattacharya fournit les conditions nécessaires et suffisantes pour casser les liens de visibilité entre le robot et la cible, ce qui permet de définir des stratégies de jeux pour les deux protagonistes, menant à un équilibre de Nash* [24].

Le problème consistant à maintenir une certaine distance avec la cible et une bonne visibilité sur celle-ci a de nombreuses applications dans le champ des jeux vidéos, où il est courant de vouloir maintenir une bonne visibilité sur un personnage dirigé par le joueur dans des environnements 3D. Le lecteur intéressé par ce domaine en trouvera une bonne introduction dans [124]. Dans ce contexte, la qualité de la navigation de la caméra est cruciale pour obtenir une bonne expérience de jeu. Bien sûr, les contraintes sont différentes de celles du monde réel : en particulier, il est possible de se « téléporter ». Il y a néanmoins des difficultés communes, et les solutions dans les jeux vidéos sont largement éprouvées par la pratique : leur efficacité peut être une source d'inspiration pour élaborer des solutions en robotique.

Enfin, il nous semble également pertinent de mentionner des problèmes moins abordés ou plus marginaux. Par exemple Bandyopadhyay et collab. étudient la poursuite furtive, dont le but est de pister visuellement une cible entre des obstacles tout en restant caché aux capteurs de celle-ci [20]. Ce problème est lié à un récent domaine d'application de la robotique appelé *covert robotics*, dont Al Marzouqi a fait une récente étude [3].

Vo et Lien, de leur côté, tâchent de tirer parti des caractéristiques de cibles « cohérentes » (foules, essaims, troupeaux, *etc.*) afin de mieux les trouver et les pister dans leur environnement [173]. La principale caractéristique d'une cible cohérente est que celle-ci a besoin d'un minimum d'espace pour pouvoir se cacher, bien que sa forme ne soit pas définie. Les cibles cohérentes sont constituées de plusieurs entités mais considérées comme un tout indissociable, d'où leur présence dans les problèmes monocibles. Avec des objectifs légèrement différents, Wang et collab. pistent une cible avec un essaim de robots [174] – ils en considèrent jusqu'à soixante en simulation. Malgré la présence de nombreux robots, cela reste un problème de suivi car l'essaim de robots est considéré comme une seule entité, bien que chacun de ses membres ait son propre comportement local. Le pistage s'effectue à travers un processus de consensus et, en contrôlant la cible, le système peut également permettre de forcer l'essaim à suivre une trajectoire prédéfinie.

RÉSUMÉ Les problèmes de suivi ont été très étudiés depuis des décennies. Ils sont souvent désignés sous les termes de problèmes de poursuite-évasion ou problèmes du Lion et de l'Homme, et de nombreuses approches ont été proposées pour les résoudre. La nature intrinsèque de ces problèmes favorise les approches géométriques, qu'elles soient locales ou globales. Il a par ailleurs été démontré que le problème est pleinement décidable, mais que le calcul d'une stratégie globale optimale est NP-complet. C'est pourquoi celles-ci ne sont pas implémentées en pratique : on leur préfère des stratégies réactives locales qui ont montré leur efficacité tant en simulation qu'à bord de vrais robots.

3.3 OBSERVATION – CMOMMT

TRAVAUX PIONNIERS Les problèmes d'observation, ou CMOMMT (voir le chapitre 1), visent à observer simultanément plusieurs cibles mobiles à l'aide d'une équipe de capteurs coopérant entre eux. Outre les déplacements des robots, il y a avant tout un aspect « allocation de tâches » – ou en l'occurrence « allocation de cibles» ; ce problème d'allocation est NP-difficile. Parker et Emmons ont proposé une approche distribuée fournissant des solutions approchées, et appelée A-CMOMMT [128]. Ils utilisent un formalisme précis nommé ALLIANCE pour le contrôle de haut niveau, tandis que les robots sont guidés à bas niveau par un champ de force. Le contrôle de haut niveau permet d'activer ou de désactiver certaines forces selon le comportement choisi pour le robot, ce qui permet d'assurer l'observation des cibles tout en évitant le problème des minimums locaux des champs de force. L'algorithme a été validé expérimentalement à l'aide d'une équipe de plateformes mobiles [127].

Jung et Sukhatme proposent de leur côté une approche basée sur une partition grossière de l'espace en plusieurs régions [78] : à l'intérieur de chaque région, les observateurs utilisent une stratégie locale permettant de maximiser le nombre de cibles observées. Il n'y a pas, ici, de réelle

phase de négociation entre les robots permettant de définir l'allocation des cibles. Les algorithmes ont été testés tant en simulation qu'à bord de robots. Deux points nous semblent importants dans les résultats présentés : d'une part les expériences avec les plateformes robotisées ont montré des performances bien moindres qu'en simulation, ce qui s'explique aisément par les plus grandes difficultés rencontrées : mauvaise localisation, champ de vision plus limité, *etc.* D'autre part les auteurs ont testé et comparé différentes compositions d'équipes de capteurs fixes et mobiles. Ils soulignent les bénéfices de tels capteurs, en particulier lorsque la coordination entre les robots est mauvaise : en effet, dans ce dernier cas, une équipe mixte voire une équipe de capteurs fixes seulement – mais « bien » positionnés – montrent de meilleures performances qu'une équipe uniquement constituée de capteurs mobiles. On peut regretter que les auteurs n'aient pas abordé le problème non trivial du positionnement des capteurs fixes – voir aussi la section 2.1 à ce sujet.

COOPÉRATION ACTIVE ET COMMUNICATIONS Plus récemment, Kolling a proposé une approche comportementale (*behaviour-based*) distribuée, appelée B-CMOMMT en référence aux travaux de Parker. L'idée est que chaque robot est responsable d'un ensemble de cibles qu'il a dans son champ d'observation, mais il est possible de demander de l'aide aux robots coéquipiers lorsque cela semble nécessaire [89]. Cette coopération active entre les robots mène à de meilleurs comportements coopératifs, comme illustré à la figure 13. Plus important encore, ce nouvel algorithme est capable, à travers les communications et la coopération active, de surmonter des situations que l'ancien système A-CMOMMT ne pouvait gérer, ceci sans perte de performances dans les autres situations. On peut néanmoins regretter que ce nouveau système n'ait pas été implémenté sur des robots, mais uniquement testé en simulation.

FIGURE 13 – En coopérant activement, les robots observateurs (en bleu,) sont capables de s'échanger les cibles (plus petites, en rouge) dont ils ont la charge, ce qui permet d'observer de meilleures performances globales – tiré de [89].

Les valeurs spectrales sont une généralisation de la notion des valeurs propres. La théorie spectrale des graphes s'intéresse aux spectres des matrices représentant les graphes.

Le système proposé par Derenick et collab. se base lui sur une discrétisation temporelle et une résolution par méthode d'optimisation [46]. Leurs travaux s'appuient sur des résultats en théorie spectrale des graphes* pour fournir des garanties de performances quant à la pleine couverture des cibles observées, mais également concernant la connectivité des robots entre eux. Malheureusement, les faibles résultats expérimentaux, obtenus en simulation *ad hoc* et dans des environnements sans obstacles, limitent la portée de cette approche.

Tang et collab. abordent eux aussi le problème d'observation comme un problème d'optimisation : considérant une équipe d'AAVs à voilure fixe, ils cherchent à minimiser le temps moyen entre deux observations consécutives de chaque cible désignée [165]. L'accent est mis sur la cinématique des agents et sur les stratégies de déplacements.

VARIATIONS ET LIENS AVEC LES AUTRES PROBLÈMES Les problèmes d'observation sont régulièrement mêlés à d'autres problèmes abordés dans notre taxonomie, essentiellement parce que les objectifs se complètent bien. Ce couplage des problèmes d'observation et de détection s'explique d'autant mieux que ces deux apparaissent généralement en séquence [127, 89], la phase d'observation suivant celle de détection.

Par exemple, dans une approche plus centrée « environnement », Pimenta et collab. abordent le problème de *Simultaneously Covering an area And Tracking intruders* (ou SCAT) [133], déjà évoqué en couverture de zones – section 2.1. Le problème d'observation est alors mêlé à celui de couverture de zones, chaque robot ayant la charge d'une sous-région et des cibles s'y trouvant, les sous-régions étant définies par la densité de cibles.

Les problèmes d'observation ont aussi de forts liens avec les autres problèmes de pistage qui ont des objectifs différents. Il est notamment intéressant de coupler les problèmes de localisation de cibles avec ceux d'observation lorsque la précision de l'estimation des positions des cibles est importante [80, 74] – voir aussi la section 3.1. Les problèmes d'observation apportent alors la composante « allocation de cibles » aux problèmes de localisation.

Enfin, tout comme en suivi, il est possible d'aborder les problèmes d'observation selon le point de vue des cibles : par exemple, Markov et Carpin définissent une stratégie d'évasion globale permettant aux cibles de coopérer activement pour échapper aux observateurs [99]. Les cibles sont notamment guidées par des règles de haut niveau telles que « rester caché aussi longtemps que possible » ou « être écartés les uns des autres tout en maintenant les liens de communications ». Ces règles ont pour but de compliquer la tâche des observateurs.

RÉSUMÉ Les problèmes d'observation (ou CMOMMT) sont des problèmes plus récents que les problèmes de suivi. Ils se sont notamment développés avec l'enthousiasme pour les problèmes et approches multirobots apparus vers la fin des années 90. Les approches proposées mettent en évidence que les équipes multirobots travaillant activement en synergie ont de meilleures performances que plusieurs instances monorobots fonctionnant en parallèle. Sans cette coopération active, les performances se dégradent fortement, et peuvent même mener à une détérioration des résultats globaux.

4

ANALYSE ET SYNTHÈSE DE L'ÉTAT DE L'ART

L'intérêt que j'ai à croire une chose
n'est pas une preuve de l'existence de cette chose.

— Voltaire

Les chapitres 2 et 3 ont présenté un état de l'art des travaux en détection et pistage de cible, articulé autour d'un taxonomie orientée sur la définition des problèmes et présentée au chapitre 1. Nous souhaitons ici proposer une analyse transverse à ces problèmes, rapprochant tous ces travaux pour mettre en lumière de manière critique et constructive les principales approches et tendances observées. Comme indiqué précédemment, il est pertinent de rapprocher les travaux des chapitres 2 et 3 car en définitive ils utilisent des plateformes, des modèles et un formalisme similaires. Ceci est tout à fait logique puisque dans la pratique, les scénarios de détection et de pistage de cibles sont fortement corrélés, et surviennent en séquence voire en parallèle.

Dans un premier temps, nous analysons les approches, considérations et hypothèses récurrentes rencontrés dans cet état de l'art présenté aux chapitres précédents. La seconde section présente les modèles sur lesquels s'appuient ces approches. Enfin, nous soulignerons l'importance du processus de validation expérimentale, tout en mettant en exergue certaines bonnes pratiques malheureusement trop rares.

La seconde partie du manuscrit s'appuie en grande partie sur les analyses présentées ici et sur le bilan qui en est fait.

4.1 APPROCHES RÉCURRENTES

La robotique étant au croisement de nombreuses disciplines, on peut trouver dans l'état de l'art une grande variété d'approches abordant des problèmes similaires, si ce n'est identiques. Parmi les nombreux critères définissant une approche, nous distinguons (i) les résultats théoriques ou analytiques des résultats expérimentaux, (ii) les systèmes centralisés des systèmes décentralisés, (iii) les comportements coopératifs des comportements « individualistes », (iv) la prise en compte des incertitudes, notamment à travers des modèles probabilistes, et (v) les processus de planification « classiques » des processus d'optimisation. On ajoutera par ailleurs que le choix d'une approche implique évidemment celui de modèles appropriés : ces derniers sont discutés à la section suivante.

4.1.1 *Théorie et Pratique*

Parmi les différents problèmes analysés dans notre état de l'art, on peut distinguer deux tendances : une grande partie des travaux se concentrent sur les résultats expérimentaux [16, 79, 20, 164] tandis que d'autres apportent des résultats théoriques [2, 7, 55, 93, 50]. Cette distinction peut paraître grossière, et plusieurs contributions contiennent ces deux types de résultats [4, 72, 179]. Nous croyons néanmoins qu'elle reste pertinente pour analyser la portée des résultats étudiés.

Les résultats théoriques sont essentiels car ils permettent de mieux orienter la recherche de solutions pratiques. Ainsi, de nombreuses contributions théoriques ont déterminé la complexité d'une classe de problèmes donnée [72, 123, 100, 90, 69, 74] : en cas de problèmes NP-difficile, il est alors pertinent de se concentrer sur la recherche de solutions hors-ligne, ou bien de solutions en ligne, mais sous-optimales. Certains résultats théoriques indiquent également la forme générale des solutions optimales, ce qui permet de restreindre les efforts de recherche ou de juger *a priori* de la qualité d'une solution. LaPaugh par exemple a démontré que la « recontamination » n'était pas utile pour résoudre les problèmes de capture [93].

"Analytical techniques have been developed for solving [problems] in complex geometrical environments. However, these previous approaches are very computationally expensive – at least exponential in the number of robots – and cannot be implemented on robots operating in real-time" – Parker [127]

Malgré tout, et mis à part ces jalons théoriques, la plupart des approches analytiques ne sont pas applicables en conditions réelles, notamment lorsqu'entrent en compte les contraintes de calculs. Il est alors nécessaire de se pencher sur des approches et des résultats plus expérimentaux, c'est-à-dire des solutions « qui fonctionnent »*.

Les travaux se concentrant sur l'intégration et la validation expérimentale ont leurs défauts, mais nous tenons à souligner l'importance de ces contributions puisque ce qui compte au final c'est d'avoir des solutions effectives aux problèmes réels. Ces approches reposent souvent sur des considérations locales, et mènent à des systèmes efficaces voire élégants [20, 117, 66]. Les algorithmes proposés sont la plupart du temps appuyés par un processus de validation expérimentale plutôt que par des garanties théoriques. Nous analysons ce processus de validation à la section 4.3.

4.1.2 *De la décentralisation*

Les problèmes rassemblés dans la taxonomie sont essentiellement des problèmes multirobots, ce qui soulève la question de la (dé)centralisation des systèmes. Pléthore d'algorithmes centralisés existent dans la littérature [122, 68, 65, 165, 153]. Les systèmes centralisés sont commodes car ils sont plus faciles à étudier et garantissent plus facilement une optimalité globale. Ils ont néanmoins leurs défauts face aux réalités du monde : ils requièrent une pleine connectivité de communication entre les agents, sont sensibles aux environnements dynamiques – surtout quand les solutions sont calculées hors-ligne – et la plupart d'entre eux subissent de plein fouet les problèmes d'échelles à cause d'une complexité exponentielle, en particulier avec le nombre de robots [106].

La plupart des problèmes relatifs aux cibles sont en effet NP-difficiles au mieux, c'est pourquoi, comme souligné dans la section 2.3, il est difficile d'attendre des performances élevées de la part des algorithmes centralisés, en particulier avec la taille grandissante des équipes de robots. Il est préférable de se concentrer, au contraire, sur les bénéfices des systèmes décentralisés [70]. Ces derniers sont en effet plus robustes, présentent une complexité raisonnable en le nombre de robots, et sont plus adaptés aux contraintes rencontrées sur le terrain (environnements dynamiques, contraintes de communications, *etc.*), ce qui facilite grandement leur intégration [107, 78, 146, 79]. Parmi les revers de la décentralisation, on compte des solutions généralement sous-optimales, présentant une optimalité locale uniquement [36, 37, 40]. Pour autant, les performances restent bonnes, en particulier dans des conditions réalistes, ce qui peut expliquer l'intérêt que les approches décentralisés suscitent actuellement [128, 171, 180, 109, 40, 91, 16].

4.1.3 *Le besoin de coopération*

Comme indiqué à de multiples reprises, la plupart des problèmes étudiés dans notre état de l'art sont des problèmes multirobots, que ce soit parce qu'il est nécessaire d'utiliser plusieurs robots pour résoudre le problèmes de manière adéquate, ou parce que la qualité des solutions bénéficie largement d'une approche multirobot et de la flexibilité qu'elle apporte. Cela posé, il existe de nombreuses façons d'organiser une équipe de robots : nous en développons les principales dans cette section.

Les robots organisés en équipe peuvent coopérer entre eux, ou simplement accomplir leurs tâches indépendamment les uns des autres, suivant une allocation antérieure de ces tâches. Ce dernier cas est très bien illustré par les schémas de patrouille en cycle ou à base de partition (voir la section 2.4). Son principal avantage est qu'il ne nécessite pas de communications une fois l'allocation initiale des tâches effectuée, ce qui élimine *de facto* les difficultés associées. En revanche, ce schéma est très sensible aux échecs des robots ou aux changements puisque les robots ne peuvent modifier les stratégies de manière globale ou coordonnée.

La coopération en ligne entre les robots permet de résoudre efficacement certains problèmes ; elle est nécessaire là où une coordination étroite est requise (capture) ou pour s'adapter dynamiquement aux aléas et observations (observation, localisation de cibles, fouille). D'une manière générale, l'impact de la coopération est fortement couplé à la structure des environnements : dans un lieu très structuré comme des bureaux, les robots bénéficient moins de la coordination qu'en environnements ouverts car ils sont « guidés » naturellement par la topologie* [13].

Nous faisons également ce constat dans nos propres travaux sur les schémas de patrouille – voir section 7.3.4.

La coopération peut par ailleurs être explicite ou implicite. Une coopération explicite permet de contrôler finement le système puisque chaque décision ou action y est discutée et diffusée à toute l'équipe, ou du moins aux membres concernés. Cela peut néanmoins engendrer une charge de

communication importante et augmente la complexité algorithmique du processus de décision.

La commande coopérative est une forme courante de coordination implicite [110] tandis que les mécanismes d'*allocation de tâches* forment la plupart des processus de coopération explicite. Il existe une vaste littérature concernant ces mécanismes et les détailler dépasse le cadre de notre analyse, mais il est intéressant de noter que les problèmes de détection de cibles sont souvent utilisés comme outil d'évaluation et de comparaison des algorithmes d'allocation. Dans notre taxonomie, ils se rapprochent le plus souvent des problèmes de chasse car ils ne cherchent pas à fournir de garantie, la détection étant alors le prétexte à l'ensemble des tâches, et non l'objectif premier.

La pertinence des mécanismes d'allocation de tâches se retrouve essentiellement dans leur caractère pleinement distribué. On peut également remarquer un plus fort taux d'intégration au sein des robots. Les stratégies de détection exhibées ne sont pas des plus impressionnantes : on observe souvent une simple recherche aléatoire effectuée en parallèle. Elles s'avèrent néanmoins robustes aux dérives, au manque d'information et aux problèmes de communication, ce qui constitue autant d'avantages indéniables. On pourra par exemple se référer au système MOVER de Jennings [77] pour de la coopération pleinement distribuée et parallèle, aux travaux de Pirjanian pour une approche multiobjectif [134], ou encore à Cai [35] ou au système HOPLITES de Kalra [79] pour des systèmes d'allocation par enchères.

4.1.4 *Environnements incertains et dynamiques*

Dans le monde réel les robots font face à de nombreuses incertitudes : les capteurs sont sujets au bruit et aux biais, les communications échouent, les comportements des agents sont difficilement estimés avec précision car le résultat même des actions est bien souvent incertain, *etc.* Ne pas tenir compte de ces incertitudes conduit généralement à un système « défectueux » en situation réelle. Afin de limiter ce phénomène, il est possible de mettre en place des stratégies différentes, qui se complètent : prendre en compte les incertitudes directement au niveau des modèles, et replanifier en ligne lorsque l'écart entre les modèles et la réalité devient trop grand.

Depuis une vingtaine d'années, la communauté des roboticiens a porté un intérêt sans cesse croissant sur les modèles probabilistes, et ce pour l'ensemble des problèmes abordés, y compris ceux étudiés ici. Ceci s'explique par le besoin d'améliorer les modèles sur lesquels raisonne la planification, et par les capacités de calculs grandissantes permettant maintenant de gérer correctement des modèles plus complexes et élaborés. Par ordre d'apparition, les principaux modèles probabilistes rencontrés en robotique et en intelligence artificielle sont les probabilités classiques [68, 171, 54, 106, 157, 164, 60], les modèles bayésiens [175, 109, 156, 138, 141, 135], les modèles markoviens (MDP) [2, 153, 122, 12], et les fameux POMDP [58, 52, 69, 21].

Ces derniers présentent de nombreux attraits mais leur coût de calcul prohibitif limite leur diffusion et leur déploiement face à des problèmes de taille raisonnable ou conséquente [152, 21]*.

De récentes avancées comme les point-based POMDP *laissent entrevoir un avenir plus radieux aux coûteux POMDP [92, 161].*

Les approches probabilistes permettent d'élaborer des stratégies plus fines mais à un coût qui reste néanmoins significatif. Elles sont par ailleurs assez sensibles à l'étape de modélisation. Yu et collab. ont par exemple montré que le choix de l'ordre des modèles markoviens a une forte influence sur la performance [178], et déterminer le meilleur choix n'est pas trivial : en particulier, augmenter l'ordre n'est pas toujours bénéfique.

Les coûts de calculs des modèles probabilistes sont d'autant plus conséquents que la plupart des problèmes abordés dans notre état de l'art sont au moins NP-difficiles. C'est pourquoi de nombreuses solutions sont calculées hors-ligne dès que la taille de l'instance devient un tant soit peu raisonnable [27, 90, 4, 106]. Malgré tout, le monde réel est rarement statique, et il est souvent nécessaire de réagir à des aléas ou des événements externes, non contrôlés (les mouvements de la cible par exemple). À cause des coûts de calculs mentionnés plus haut, il n'est pas envisageable de précalculer un ensemble de stratégies pour chaque état possible du monde permettant de faire face à ce dynamisme et ces aléas. Il est alors nécessaire de calculer ou mettre à jour ces stratégies en ligne, au cours de la mission.

L'apport en robustesse est réel car il est alors possible de s'adapter à la demande à toute situation nouvelle et inattendue. Ces calculs en ligne ne se font cependant pas gratuitement : ils nécessitent bien souvent, pour réagir en temps fini, de ne faire que des considérations locales et donc d'accepter le caractère sous-optimal de la solution calculée. Ce compromis, qui est aussi un choix cornélien, est bien illustré par la section 3.2 de l'état de l'art : il a été démontré que le suivi de cible est entièrement décidable, mais ce problème de décision est NP-complet [116]. En conséquence, l'état de l'art est constitué d'approches locales calculées en ligne [117, 18] ; bien que sous-optimales, elles s'avèrent très performantes, même dans des conditions difficiles.

4.1.5 *Planification et Optimisation*

Nous avons analysé aux chapitres 2 et 3 un ensemble de problèmes de détection et de pistage de cible du point de vue de la décision, dont le but est très grossièrement de déterminer « qui fait quoi, où et quand ? » En très grande majorité, les travaux présentés formulent les problèmes soit comme des problèmes de planification, soit comme des problèmes d'optimisation. Bien que les problèmes sous-jacents soient identiques, les formulations diffèrent largement, de même que les solutions et les résultats.

*En planification, il est courant de décrire le problème à résoudre à l'aide du langage PDDL (*Planning Domain Definition Language*) ou de ses variantes. La description est sémantique.*

En planification, on cherche à simuler des actions et leurs effets, afin de décider quelle séquence d'actions nous permet d'atteindre un état objectif à partir d'un état initial (voir aussi la section 5.2). On peut trouver de nombreux formalismes permettant d'implémenter ces planificateurs*, mais la phase critique reste toujours la modélisation des espaces d'actions, c'est-

à-dire de l'état du monde et des effets des actions sur ceux-ci. Une fois proprement modélisés, de nombreux planificateurs permettent de résoudre le problèmes et fournissent une solution sur la base de ces modèles.

En optimisation, le problème est décrit par un ensemble de fonctions et de valeurs, par exemple comme à l'équation (11) p.104. La description est valuée.

En optimisation, on définit une fonction valuée appelée « fonction objectif » que l'on cherche à optimiser, c'est-à-dire généralement à minimiser ou maximiser, tout en respectant un ensemble de contraintes définies mathématiquement*. L'étape critique consiste, ici, à valuer l'état du monde et les actions, et à définir une fonction objectif adéquate. Il suffit ensuite de choisir un solveur parmi les nombreux disponibles afin d'obtenir une solution – voir aussi la section 2.4.

La différence fondamentale entre optimisation et planification réside donc dans le fait que la première travaille sur des valeurs numériques, tandis que la seconde travaille plus qualitativement. Évaluer correctement ces valeurs numériques peut être assez direct et évident – par exemple lorsque l'on tente de minimiser l'« oisiveté » en patrouille. Cela peut aussi s'avérer très complexe et sensible dans certains cas (comme en capture). L'optimisation est proche des données brutes et du contrôle, au contraire de la planification qui demande une expressivité sémantique mais permet un haut niveau d'abstraction dans les modèles. Il est alors possible de faire de la planification hiérarchique, dans laquelle les tâches peuvent être décomposées en sous-tâches ce qui permet de gérer la complexité du problème en jouant sur sa granularité – voir à cet effet les *Hierarchical Task Networks (HTN)* [56].

Parmi les différentes méthodes d'optimisation stochastique, on compte notamment les méthodes de Monte-Carlo, le recuit simulé, l'entropie croisée, les algorithmes évolutionnaires, les algorithmes en essaim, etc.

À l'opposé, et plus récemment, l'utilisation de techniques d'optimisation stochastique* a permis de résoudre de manière efficace des problèmes complexes comme le TSP en s'appuyant sur les capacités de calculs actuelles des ordinateurs. Ce type de méthodes s'appuient sur une exploration aléatoire « guidée » de l'espace de recherche des solutions afin de trouver une solution valide proche de l'optimal. Ces différentes méthodes empiriques sont autant de métaheuristiques permettant de se rapprocher de la solution optimale.

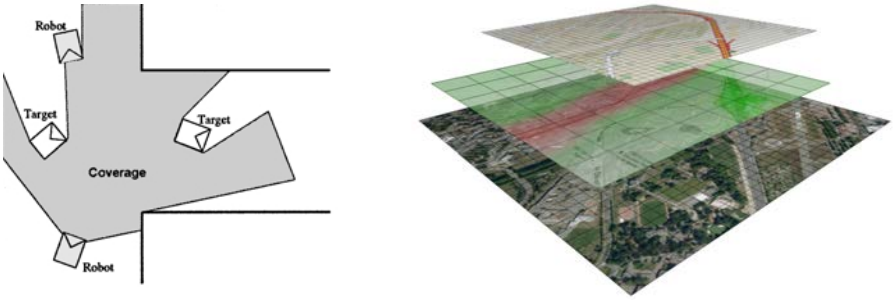
4.2 Modèles principalement exploités

Le lecteur aura sans doute remarqué que la plupart des modèles rencontrés dans les chapitres 2 et 3 sont plutôt simples, voire beaucoup trop simples à notre avis. Leur expressivité est limitée, et s'ils facilitent la résolution du problème en réduisant sa complexité d'une manière ou d'une autre, ils n'offrent qu'une représentation réductrice de la réalité, ce qui rend leur intégration difficile et limite le réalisme du processus de validation de ces approches. Nous présentons dans cette section les principaux modèles rencontrés tout au long de l'état de l'art. Nous distinguons ici les modèles d'environnement et les modèles des agents, bien que le choix des uns et des autres ne puissent être fait indépendamment, la structure de l'environnement influençant nécessairement les modèles des actions. Nous proposons par ailleurs notre propre vision des modèles en robotique au chapitre 5.

4.2.1 *Modèles de l'environnement*

AU-DELÀ DES MODÈLES 2D La plupart des auteurs utilisent une représentation 2D et unique (« monocouche* ») du monde, que ce soit sous forme de grille [178, 59, 148] ou de modèles plus continus : triangulation [57], décomposition de Voronoi [133], décomposition spécifique [28, 113] ou sans décomposition aucune [117, 18, 81]. En conséquence, les obstacles au mouvement et à la perception sont confondus, et des situations réalistes ne peuvent fréquemment pas être représentées, comme les régions à travers lesquelles les AGVs peuvent voir mais ne peuvent traverser (fossés, étendues d'eau) ou que les AAVs peuvent survoler mais ne peuvent observer (intérieur des bâtiments, souterrains).

Un modèle monocouche utilise une représentation unique de l'environnement (hauteur ou obstacle par exemple), limitant de facto *sa richesse d'expression – figure 14 (a).*



(a) environnement monocouche (b) environnement multicouche

FIGURE 14 – Différents modèles d'environnement : en (a) les obstacles au mouvement et à la vision sont confondus, et communs à l'ensemble des agents (tiré de [78]) ; en (b) les différentes résolutions et types de données peuvent coexister (tiré de [32]).

Un modèle multicouche distingue plusieurs niveaux de sémantique (mouvement, communication, capteurs, etc.*), ce qui enrichit son expressivité – figure 14 (b).*

Avec la puissance de calcul sans cesse grandissante, des modèles plus complexes (2.5D [88], 3D [22], ou multicouches* [59, 148]) ont récemment été proposés. De tels modèles sont capables de contenir et d'exprimer plus d'informations, étant par là même plus réalistes et permettant des stratégies plus fines, potentiellement meilleures. Il est par exemple possible de tirer avantage de postes d'observation situés en hauteur pour obtenir un point de vue plus large [88] ou de distinguer les obstacles aux mouvements de ceux qui bloquent les capteurs [148].

Il est par ailleurs possible d'améliorer les modèles à l'aide des nombreuses cartes facilement accessibles de nos jours. Flushing *et al.*, par exemple, planifient des missions de secours en milieu naturel (*Wilderness Search And Rescue* (WiSAR)) pour des équipes hétérogènes constituées d'une large diversité d'agents aux capacités cognitives et sensorimotrices variées : humains, animaux, robots, *etc.* [59]. Leur approche repose sur l'utilisation de systèmes d'informations géographiques : ceux-ci fournissent des modèles de précision permettant la prise en compte de l'influence des caractéristiques du terrain et des conditions environnementales sur les performances des différents agents, afin de répondre au mieux aux situations d'urgence.

À côté de ces modèles plus réalistes, certains auteurs exploitent au contraire des représentations très abstraites telles que les espaces CAT(k) [8, 7]. Ces espaces abstraits facilitent la recherche de solutions, et permettent d'évaluer rigoureusement la complexité algorithmique. Néanmoins, la perte d'information induite par le niveau d'abstraction peut restreindre la validité des solutions exposées lorsque celles-ci sont confrontées au monde réel.

MONDES DISCRETS Les grilles et les décompositions discrètes – ou maillages – sont de loin les modèles les plus utilisés. Ils permettent en effet de définir facilement une structure de graphe, ce qui permet de développer des algorithmes s'appuyant sur la théorie des graphes pour laquelle il existe une littérature abondante [129, 100, 113, 130, 46]. Pour autant, transformer un modèle métrique continu du monde en un modèle topologique cohérent et faisant sens n'est pas une tâche facile [106]. Seuls quelques auteurs proposent explicitement un moyen d'y parvenir [71, 87, 97] ; d'autres présupposent que le graphe est déjà disponible, qu'il soit construit à la main ou par un processus antérieur [69], ou utilisent un échantillonnage plus ou moins aléatoire comme un compromis entre un modèle continu et modèle discret.

"While discretization invariably implies a certain level of approximation and deviation from the original metric space, in order to make any continuous problem computationally feasible it is an indispensable trade-off." – Bhattacharya [27]

La discrétisation du monde permet aussi de contourner la complexité algorithmique des modèles numériques continus*, même quand les algorithmes sont théoriquement compatibles avec ces modèles continus [162]. En fait, on constate que les modèles continus sont surtout utilisés en commande, pour un contrôle réactif local, ou par des processus de décisions gloutons [117, 157, 18]. À l'inverse, les modèles discrets sont également utilisés pour des preuves mathématiques dans des représentations abstraites [129, 121, 8].

Enfin, si l'hypothèse de continuité est séduisante et facilite les démonstrations et les garanties formelles, elle soulève un autre problème : on rencontre en effet de nombreuses discontinuités intrinsèques au monde réel qui impactent tant les mouvements que la vision, et que les modèles continus ne peuvent représenter correctement [75].

LA RELATION ESPACE-TEMPS En plus de la représentation de l'espace, la représentation du temps s'avère cruciale et, en réalité, peine à être décorrélée de la première, notamment lorsque les modèles sont discrets. On notera qu'il est tout à fait possible de considérer un temps discret pour un modèle continu de l'espace [18] mais les représentations continues du temps sont aussi utilisées [8]. La discrétisation temporelle est typiquement utilisée pour définir un nombre fini et dénombrable d'états du monde. Hélas, l'approximation induite va probablement réduire la cohérence entre l'état réel du monde et son modèle*. Il est alors nécessaire de s'assurer que les algorithmes restent robustes à de telles incohérences.

"The next state x_{k+1} will usually not lie exactly at a discretized value" [be it temporal or spatial]. – LaValle [94]

Afin de faciliter les preuves mathématiques, l'hypothèse « une unité spatiale parcourue par unité de temps » s'avère commode et ne mène souvent à aucune perte de généralité [162]. Elle peut malgré tout avoir un impact

sur la qualité des solutions résultantes. En fait, la relation entre l'espace et le temps se révèle particulièrement sensible quand une coordination précise est exigée, ou qu'un robot ne peut « attendre » ses coéquipiers, par exemple lors de la poursuite d'une cible [148, 164]. Dans les deux cas, l'expérimentation validera ou non les hypothèses sur le temps, et c'est pourquoi il est important de mener cette étape de validation avec soin et rigueur – voir la section 4.3.

ENVIRONNEMENTS PARTICULIERS Certains environnements particuliers peuvent remettre en question certaines hypothèses sur lesquelles s'appuient les algorithmes, voire contraindre ou prévenir l'exécution de certaines actions. Par exemple, certains algorithmes présupposent qu'il est possible de « marquer » l'environnement, d'une manière ou d'une autre [40, 64]. Il est nécessaire de porter la plus grande attention au caractère réaliste ou non de cette hypothèse de marquage, et en particulier à son coût matériel.

Les récentes avancées de la technologie RFID offrent une solution technique intéressante au marquage de l'environnement [98] mais celle-ci ne peut s'appliquer dans tous les contextes. C'est une difficulté malheureusement masquée par la plupart des simulateurs dans lesquels il est très facile d'émuler ce marquage*. Seule une validation expérimentale réaliste pourra valider l'approche – voir la section 4.3. Pugh et Martiloni, par exemple, ont étudié l'intégration d'un algorithme par essaim (*swarm algorithm*) en une stratégie de recherche mulitrobot, et en particulier l'impact de la distance de détection des puces RFID permettant d'émuler le marquage dans le monde physique [140]. Dans leur cas, la perte de performances de l'algorithme est restée faible face à des environnements de taille limitée.

"Robotic systems come with their own hypotheses that are more restrictive than in simulation." – Glad *et al.* [64]

Alors que la plupart des travaux de l'état de l'art abordent des environnements simples, et plus spécifiquement des environnements à deux dimensions avec une catégorisation binaire « libre » ou « obstacle », certains auteurs abordent des environnements plus complexes. Par exemple, Ehlers étudie les environnements maritimes pour des missions de recherche (fouille) à l'aide de sonars multistatiques [52]. Le contexte (guerre anti-sous-marins) tout comme les propriétés intrinsèques de l'environnement (l'eau) affectent les robots en limitant leur champ d'action et leur efficacité, ce qui impose d'adapter les algorithmes habituellement utilisés.

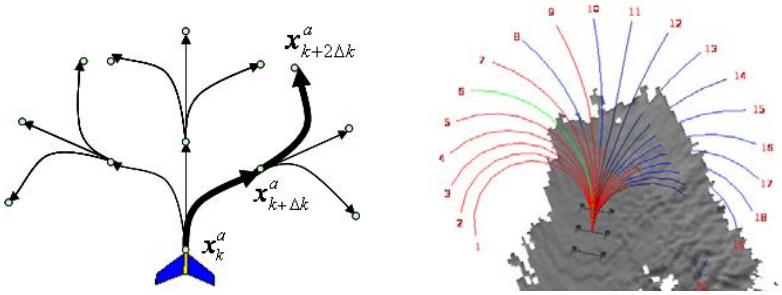
4.2.2 *Modèles des agents*

Le terme d'« agent » réunit ici les robots et les cibles, c'est-à-dire tous les protagonistes des scénarios considérés dans notre taxonomie – voir les chapitres 2 et 3. Dans ce même contexte, les actions considérées sont les mouvements, les observations et les communications. Chacune de ces capacités nécessite son propre modèle, et le choix de ce modèle est fortement lié à celui de la représentation de l'environnement, les deux étant « convolués » pour prédire le résultat des actions envisagées – voir section 5.2.

MODÈLES DE MOUVEMENT On trouve dans la littérature de nombreux modèles de mouvement, dont la précision et la justesse varient grandement, en particulier concernant leur capacité à rendre compte les contraintes cinématiques et dynamiques des systèmes. Plus la représentation du monde est abstraite, plus le modèle de mouvement l'est aussi. Ainsi, avec une modélisation basée sur des graphes, le modèle de mouvement ne décrit généralement que l'accessibilité des nœuds du graphe (i. e. d'une zone) éventuellement associée à un coût (distance, temps, énergie, *etc.*). Il est alors généralement admis que l'estimation de ce coût est correcte, et l'on se désintéresse de la manière dont le robot effectuera réellement ce mouvement. Nous pensons que cette hypothèse est parfois présomptueuse, et seule une validation correcte des algorithmes peut garantir son réalisme*.

Nous revenons sur ces aspects aux chapitres 5, 7 et 9.

Depuis quelques années, on observe fréquemment des modèles de mouvement plus précis, basés sur des primitives de mouvement précalculées (*pattern-based motion models*) pour des AAVs (à voilure fixe ou non) [165, 167] mais aussi, plus rarement, pour des AGVs [178] – voir aussi figure 15. Il existe également d'autres modèles du caractère non holonome des AGVs [118]. La prise en compte de la géométrie de l'environnement permet également d'affiner le mouvement des agents [59].



(a) (b)

FIGURE 15 – Exemples de modèles de mouvement basés sur des primitives pour (a) un AAV à voilure fixe – tiré de [125] – et (b) un AGV non holonome.

Un modèle précis ou réaliste n'est toutefois pas toujours nécessaire ni désiré, en particulier pour la cible. Ainsi, associer une vitesse infinie à la cible permet de définir des stratégies conservatives pessimistes considérant le pire cas (*worst-case guarantee*) [41].

MODÈLE DE CAPTEUR Les modèles de capteurs sont principalement utilisés pour les robots, même si c'est parfois le cas de la cible également, comme en poursuite furtive (*stealth tracking*) [20] ou lorsque le point de vue du poursuivi est considéré [99]. La plupart des modèles de capteur sont très basiques : souvent seule la distance ou bien une topologie grossière est prise en compte, en particulier quand l'environnement est modélisé par un graphe. Certains auteurs s'attachent néanmoins à considérer explicitement le champ de vue et les contraintes de visibilité [49, 61, 117, 28, 173], prenant

en compte l'orientation du capteur et l'angle de perception de celui-ci, en plus de la distance. Des modèles plus fins peuvent aussi intégrer des informations de hauteur en 2.5D ou 3D [88, 22], voire de luminosité dans une représentation multicouche du monde [148] – voir aussi figure 16.

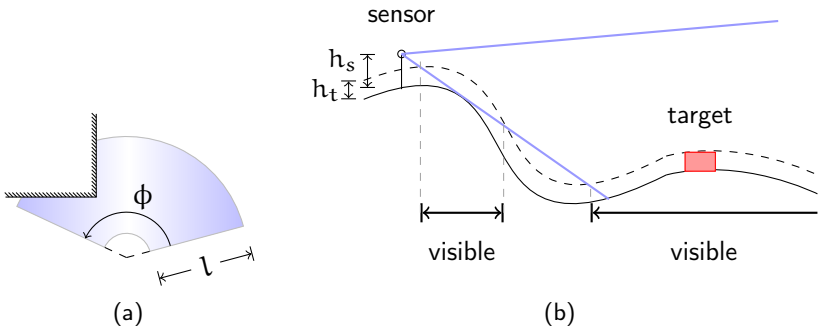


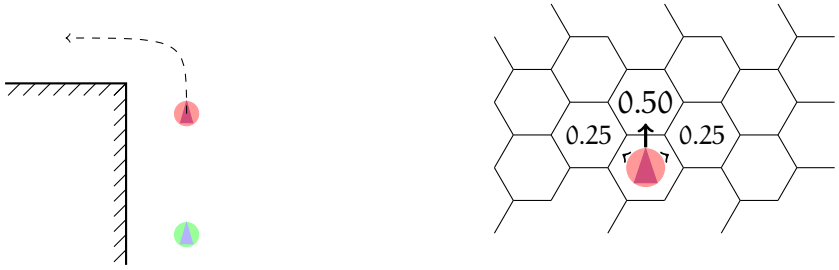
FIGURE 16 – Exemples de modèles de capteurs réalistes : (a) l'orientation, l'angle de vue et la portée du capteur son pris en compte ; (b) un lien de visibilité s'établit en ligne droite, et prend en considération la hauteur du capteur, celle de la cible recherchée, et la topographie du terrain.

La plupart de capteurs utilisés sont basés sur les ondes lumineuses, qu'ils soient des caméras ou des LIDAR, à l'exception des véhicules sous-marins autonomes (AUVs) pour lesquels les sonars sont préférés [52]. La plupart des contraintes de visibilité sont donc des obstacles à la vision, bloquant les rayons de lumière en ligne droite.

MODÈLES DE COMMUNICATION Bien que l'on puisse intuitivement considérer les communications comme étant très similaires à la visibilité (quand il existe un lien visuel, on peut raisonnablement s'attendre à ce qu'il y ait aussi un lien de communication), la plupart des auteurs font l'hypothèse d'une parfaite connectivité entre les robots (*full connectivity assumption*). Cette hypothèse est très commode mais il n'est pas difficile de remettre en question son réalisme, ce qui peut fortement détériorer l'efficacité de l'approche retenue. Il est en fait complexe et coûteux de mettre en œuvre des modèles précis pour les communications [108], et peu d'auteurs cherchent à s'en préoccuper. Ceux qui le font mettent par exemple les besoins et contraintes de communications au cœur de leurs approches, dès la conception [72], ou étudient spécifiquement leur impact [70, 140].

MODÈLES DE COMPORTEMENT En plus des modèles d'actions (mouvement, observation, communication), il peut être intéressant de modéliser le comportement attendu des cibles ou des coéquipiers, comme illustré à la figure 17. Le déplacement des cibles est souvent modélisé de manière probabiliste, que ce soit par une marche aléatoire [164, 127, 70] ou par des modèles plus élaborés : bayésien ou markovien [178, 69, 21]. Prédire le comportement et les déplacements des cibles permet d'élaborer des stra-

tégies plus sophistiquées ou moins conservatives. En outre, dans le cas où les cibles sont antagonistes, le comportement de ces dernières peut être modélisé précisément par la théorie des jeux [113, 116, 172].



(a) (b)

FIGURE 17 – Exemples de modèles de comportement pour les cibles : (a) modèle déterministe de cible évasive (cherchant à se cacher) ; (b) modèle probabiliste.

Il est également possible d'utiliser un ensemble de modèles de cibles comme une métrique pour comparer différents algorithmes [155]. Il faut aussi noter que si la modélisation du comportement des cibles est importante et permet de définir des stratégies plus fines en fonction des réactions attendues, elle peut aussi s'avérer contre-productive dans le cas où le modèle utilisé n'est pas correct [178].

Enfin, modéliser le comportement de ses coéquipiers est également pertinent : la principale motivation est de réduire le besoin de communication, dont le volume requis peut s'avérer problématique en cas de coordination explicite au sein de l'équipe. Il est en effet possible de se coordonner de manière implicite sur la base du comportement attendu de ses coéquipiers [66]. Les modèles probabilistes retenus peuvent être écrits en dur au préalable (bayésiens, markoviens) ou bien appris [177].

4.3 RÉSULTATS ET VALIDATIONS EXPÉRIMENTALES

Hormis certains résultats purement théoriques, la plupart des travaux proposent un ou plusieurs algorithmes et les modèles associés pour résoudre un problème donné. Ces algorithmes peuvent être accompagnés de garanties théoriques (complexité, optimalité, complétude) mais il est communément admis qu'il faut fournir une validation expérimentale afin de montrer que ces algorithmes sont effectivement applicables et fonctionnent réellement en pratique.

Ainsi peut-on évoquer Eaton et Zadeh qui, en 1962, ont « résolu » le problème de poursuite [50]. Aussi élégants que soient leur approche et leurs modèles, leur solution n'est pas applicable en pratique car elle pose plus de problèmes qu'elle n'en résout – elle se base notamment sur des fonctions mathématiques qu'il est difficile d'expliciter. Ceci explique pourquoi, à notre connaissance, il n'existe pas d'implémentation de leur approche à bord de

robots, et pourquoi le problème de suivi a depuis fait l'objet de nombreux travaux. Les approches actuelles, qui montrent de bonnes performances sur le terrain, sont très éloignées de la solution d'Eaton et Zadeh.

Ceci illustre très bien ce besoin de validation expérimentale : seuls les tests des algorithmes embarqués à bord de vrais robots dans des contextes pratiques et réalistes permettent de montrer la validité réelle des algorithmes, et leur capacité à gérer les difficultés du monde réel : incertitudes, non-déterminisme, systèmes asynchrones, *etc.* Mettre de côté ces problèmes courants conduit bien souvent à élaborer des algorithmes qui ne sont pas robustes, et donc peu utiles. Ceci fait d'ailleurs écho à la critique par Brooks de planification traditionnelle et des modèles (trop) abstraits, au début des années 90 [33].

Pourtant, et bien que ce constat ne soit pas récent, on observe un manque de réelle validation expérimentale dans la plupart des articles. Bien sûr les auteurs fournissent des protocoles expérimentaux (descriptions, paramètres, nombre de tests, *etc.*) et les résultats associés, mais très peu d'entre eux considèrent des conditions de tests réalistes. Ainsi la plupart des approches sont-elles « validées » par des simulations *ad hoc* [165, 157, 88, 49, 74, 103, 180, 40, 173, 12], c'est-à-dire des simulations qui négligent la prise en compte des caractéristiques du monde réel. En effet, ces mondes simulés sont bien souvent naturellement discrets en temps et en espace par construction, et ne comportent aucune incertitude, ou très peu.

Dans ce contexte, les hypothèses et simplifications initiales sur lesquelles reposent les algorithmes sont directement intégrées dans le simulateur, qui ne peut alors que rarement révéler les défauts des solutions résultantes. Il est également possible d'introduire ces mêmes limitations lors d'expériences menées avec des robots lorsque les conditions de ces dernières sont déformées pour convenir artificiellement aux modèles, par exemple en restreignant la liberté de mouvement des agents à des grilles [178, 138] ou en forçant une discrétisation temporelle [164]. Ces problèmes d'écarts entre les modèles et la réalité ont déjà été mis en exergue par plusieurs auteurs [78, 133, 64, 138].

Par ailleurs, le processus de validation expérimentale présente souvent d'autres défauts parmi lesquels le manque de comparaison avec l'état de l'art et le manque de « culture des statistiques ». Ainsi les algorithmes sont-ils souvent comparés avec des solutions triviales telles que des marcheurs aléatoires ou des solutions gloutonnes ou force brute [74, 136, 103, 75] et les résultats présentés sont rarement significatifs statistiquement parlant. Ces défauts très communs sont compréhensibles : une validation expérimentale rigoureuse est coûteuse et demande un grand travail d'ingénierie, de nombreux tests, et une logistique hors de portée de la plupart des chercheurs. De plus, les comparaisons équitables avec l'état de l'art sont limitées par la variété des robots et des environnements de tests qui existent au sein des équipes et laboratoires.

Néanmoins, certains auteurs font l'effort de fournir de bonnes comparaisons avec l'état de l'art [180, 40, 99, 173]. Plus récemment, d'autres ont

Dans un test statistique, la valeur-p est la probabilité d'obtenir le même résultat au test si l'hypothèse nulle était vraie, c'est-à-dire sans prendre en compte l'hypothèse (les explications) testée.

introduit de bonnes pratiques sur le plan des analyses statistiques [155, 13], inspirés par les pratiques d'autres communautés comme la physique, la biologie ou les sciences sociales. On y retrouve notamment l'analyse critique des écarts-types et des valeur-p*.

Des efforts ont également été fournis par une partie de la communauté pour homogénéiser les plateformes de tests ou pour réutiliser des environnements de tests déjà en usage afin de comparer au plus juste les algorithmes entre eux [164]. Les compétitions comme la RoboCup, dont la *Rescue Virtual Robot Competition* [17, 76], sont également un bon moyen de comparer plus équitablement les approches. Certains articles proposent aussi des *benchmarks* pour un ensemble d'algorithmes évalués et comparés selon différentes métriques, afin de limiter le biais induit par l'usage d'une seule métrique [76, 155], ce qui permet de fixer certaines bases et références dans les domaines considérés. Rendre le code public*– lorsque cela est possible – est également un bon moyen d'encourager la reproductibilité des résultats et les comparaisons entre équipes – voir [11] entre autres.

Nous croyons fermement que l'utilisation de simulateurs réalistes dans le processus expérimental est une des clefs permettant de gérer les problèmes de validation. En effet, ils fournissent une base commune d'environnements de tests avec des conditions réalistes et facilement accessibles, tout en permettant la mise en place de nombreux tests afin d'obtenir des résultats significatifs sur le plan statistique. Ce que nous appelons « simulateur réaliste » diffère d'un simulateur *ad hoc* dans la manière dont il modélise la réalité : le temps et l'espace simulés sont continus, et il embarque un moteur physique permettant de mieux rendre compte des interactions avec l'environnement. Le développement de tels simulateurs demande de gros efforts d'ingénierie mais heureusement il existe déjà plusieurs alternatives *open-source* disponibles gratuitement, parmi lesquelles Morse [51], Gazebo [86] et USARSim [17]. Ces simulateurs sont fournis avec un ensemble de modèles de robots et d'environnements et fournissent un cadre commun et une architecture modulaire – par exemple basé sur [142].

L'ensemble des codes sources exploités dans nos travaux de la partie II est disponible publiquement (les adresses sont indiquées aux sections concernées).

On peut alors imaginer de combiner différents algorithmes (par exemple un algorithme de planification utilisant dans ses tests un algorithme de *Simultaneous Localization And Mapping* (SLAM) développé par une autre équipe), voire de jouer différentes stratégies les unes contre les autres (par exemple une stratégie d'évasion face à une stratégie de poursuite élaborée par une autre équipe). Un processus de validation impliquant des scénarios, des modèles et du code *open-source** intégrés à un simulateur réaliste fournit *de facto* des résultats facilement reproductibles et statistiquement significatifs, tout comme une base de comparaison utile pour de futures recherches sur le domaine considéré. Cela permettrait de rassembler et mutualiser les efforts de recherche, ce qui ne ne peut que bénéficier à la communauté scientifique et aux qualités des recherches.

The most damaging phrase in the language is :
"We've always done it this way !"

— Grace M. Hopper

Cette première partie présente une taxonomie unificatrice des problèmes relatifs à la gestion de cibles en robotique (détection et pistage). Notre état de l'art va au-delà des frontières de communautés spécifiques ou de problèmes spécifiques, et élargit le champ d'étude des précédentes analyses disponibles.

Nous avons identifié les modèles et approches récurrentes et transverses aux différents problèmes étudiés. De cette analyse globale, nous tirons la conviction qu'il y a trois grandes directions dans lesquelles la communauté doit poursuivre voire accentuer ses efforts de recherche : les modèles utilisés, le développement d'approches décentralisées, et le processus de validation expérimentale.

VERS DES MODÈLES PLUS RICHES Les modèles les plus fins récemment proposés rassemblent des considérations probabilistes, des modèles multicouches de l'environnement et des représentations hiérarchiques. La complexité induite par ces modèles améliorés est maintenant abordable avec les capacités de calculs actuelles. Ces modèles répondent par ailleurs au besoin de combler ou du moins de réduire l'écart entre les modèles et la réalité. Les premiers résultats obtenus sont prometteurs et l'on peut espérer qu'ils deviennent la norme dans un avenir proche. Les communications restent néanmoins peu modélisées, et encore plus rarement de manière réaliste, ce qui limite la portée des approches proposées et empêche leur utilisation sur le terrain en conditions réelles. Nous avons par ailleurs argumenté en faveur d'une séparation claire des modèles et des algorithmes, permettant de faire évoluer les uns et les autres de manière plus indépendante, tout en encourageant l'utilisation de modèles plus riches dans le processus de décision.

VERS DES ALGORITHMES DÉCENTRALISÉS Considérant la proportion de problèmes de décisions au moins NP-difficiles en robotique mobile, nous pensons que des approches sous-optimales mais efficaces et capables de tourner en ligne à bord des robots constituent la clef permettant de résoudre les cas pratiques rencontrés sur le terrain, en faisant face aux incertitudes et aux aléas. Nous encourageons donc le développement d'algorithmes décentralisés qui ont pour autre avantage de bien vivre le passage à l'échelle. Les algorithmes dits *anytime* sont une autre alternative reliant l'efficacité et l'optimalité [71], en particulier dans les applications où le

temps est crucial comme les scénarios de *search and rescue*. Malgré tout, il est frappant de constater que la plupart des bornes sur les performances sont seulement observées expérimentalement, et il y a un manque certain de garanties théoriques sur la sous-optimalité des solutions attendues.

VERS UNE MEILLEURE VALIDATION EXPÉRIMENTALE Enfin, nous considérons qu'il est nécessaire d'apporter de gros changements dans le processus de validation des approches proposées. En effet, la robotique est une discipline résolument ancrée dans le monde réel et dans les solutions qu'elle peut apporter aux problèmes qu'on y rencontre : il est alors désirable de fournir des algorithmes qui traitent ce monde réel et fournissent des solutions applicables. Pourtant on rencontre bien trop de travaux travaillant à partir d'hypothèses irréalistes et dont la validation expérimentale contient trop de lacunes pour entrevoir cette connexion avec le monde réel. Une validation expérimentale rigoureuse devrait impliquer des résultats facilement reproductibles, une analyse statistique raisonnable et une comparaison équitable avec d'autres solutions tirées de l'état de l'art. Nous sommes conscients que cela est coûteux, mais nous pensons que ceci peut se mettre en place à moindre coût pour la communauté dans son ensemble avec l'utilisation d'architectures modulaires combinées à des simulateurs réalistes et des bases de données communes. Cela faciliterait le développement de nouveaux algorithmes, l'arrivée de nouveaux acteurs dans la recherche sur le domaine, et l'adoption des percées significatives entre les communautés, tout en consolidant la validité des résultats avancés.

"It is the authors' belief that research in this field should be more oriented towards effective solutions with applicability in the real world" – Portugal and Rocha [138].

Dans la partie II, nous abordons plus spécifiquement certains problèmes étudiés ici : la patrouille et le suivi. Nous tâchons d'y mettre en œuvre les recommandations formulées ici.

Deuxième partie

CONTRIBUTIONS ALGORITHMIQUES

Cette seconde partie s'appuie sur nos considérations faites à la partie I et commence par aborder la relation entre la planification et les modèles à partir desquelles celle-ci fonctionne. Dans l'optique d'exploiter et d'intégrer des modèles réalistes dans les processus de planification, nous présentons d'abord une organisation des modèles ainsi qu'un formalisme associé (chapitre 5). Les chapitres suivants appliquent ces principes aux problèmes de la détection de cibles (chapitres 6, 7 et 8) et du pistage de cibles (chapitre 9). Le chapitre 6 développe une nouvelle approche de planification de trajectoires de patrouille en contexte antagoniste ; les résultats expérimentaux associés sont rassemblés dans les chapitres 7 et 8. Le chapitre 9 présente lui une nouvelle approche des problèmes de suivi de cible, dans un contexte d'économie des ressources.

5 MODÈLES

Essentially, all models are wrong, but some are useful.

— George E. P. Box

Les modèles représentant l'environnement et les agents (les robots et les cibles) sont au cœur des processus décisionnels*. Ils sont nécessaires quelle que soit l'approche retenue ou le problème abordé. La combinaison des modèles d'environnement avec ceux des agents permet de prédire le résultat des actions des agents, c'est-à-dire ce que les agents sont capables de faire et les conséquences probables des différentes actions qui leur sont possibles.

Les processus décisionnels regroupent la planification et la supervision.

Ce chapitre s'appuie sur nos analyses précédentes concernant les modèles exploités dans la littérature – section 4.2. Nous illustrons d'abord l'impact des modèles sur la planification, avant de développer la notion de « modèles » et d'exprimer ce que la planification attend de ces modèles. Nous proposons ensuite une organisation générale des modèles et données telle que nous l'avons développée et souhaitons l'intégrer à bord de nos robots. Ce chapitre s'achève en décrivant l'intégration que nous en avons fait.

5.1 IMPACT DES MODÈLES

Cette première section illustre l'impact que peuvent avoir les modèles sous-jacents sur la planification. Pour cela, imaginons qu'un robot situé en p_1 souhaite observer trois positions A, B et C aux coordonnées connues, tout en cherchant à minimiser la distance parcourue – figure 18) Pour observer une position objectif, il est nécessaire de respecter une certaine distance d'observation maximale par rapport au point à observer, tout en prenant en compte l'angle de vision du capteur.

Notre robot élabore un plan constitué de trois points de passage p_2, p_3 et p_4 qu'il sait accessibles. Ce plan spécifie que le robot observera A depuis p_2, B depuis p_3 et C depuis p_4. Chaque point d'observation est le point de la trajectoire le plus proche du point à observer auquel il est rattaché. Nous étudions dans cette section les caractéristiques de ces points et de la trajectoire résultante en fonction des modèles considérés.

Les figures 18 et 19 montrent deux trajectoires correspondant à deux distances d'observation minimales différentes, avec un angle de vue à 360° et sans contrainte sur le mouvement. On remarque que la distance d'observation la plus courte contraint la trajectoire à « zigzager » d'un objectif à l'autre (figure 19), alors qu'une contrainte plus lâche permet une solution

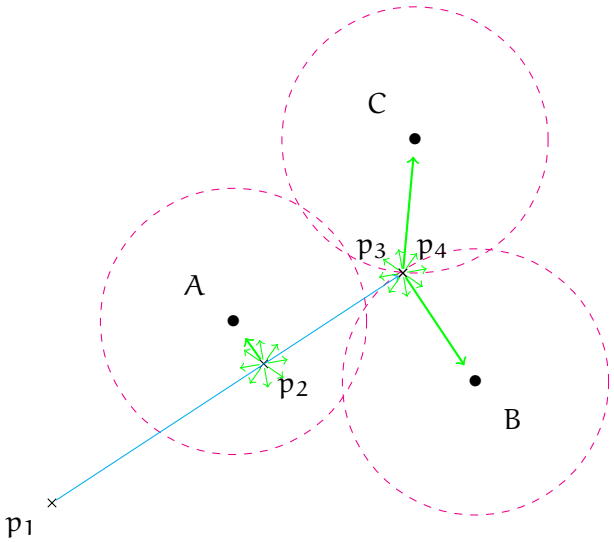


FIGURE 18 – Exemple de trajectoire planifiée à partir avec des modèles simples : un robot situé en p_1 désire observer les points A, B et C. La distance d'observation maximale est indiquée par les cercles en pointillés. La trajectoire choisie est la ligne droite passant par $(p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4)$, dans laquelle il observera A, B et C respectivement depuis p_2, p_3 et p_4. Les flèches vertes indiquent les liens de visibilité.

en ligne droite (figure 18).

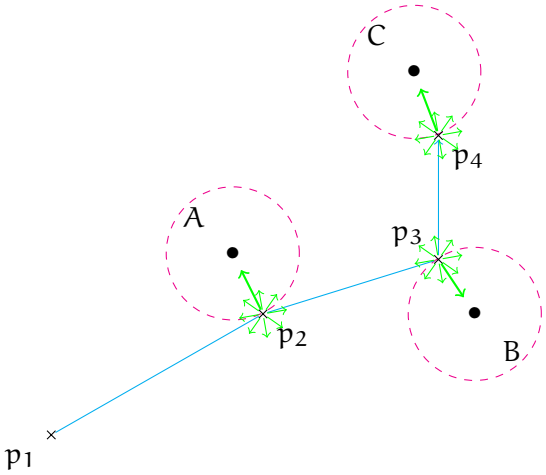


FIGURE 19 – Exemple de trajectoire contrainte par une distance minimale d'observation : cet exemple reprend celui de la figure 18 mais avec une distance minimale d'observation plus courte. Cela contraint la trajectoire $(p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4)$, qui devient une ligne brisée.

Imaginons maintenant que le planificateur raisonne spatialement sur une grille, et que celle-ci indique la présence d'obstacles : la nouvelle trajectoire

$(p_1 \to p_2 \to p_3 \to p_4)$ du robot est alors modifiée comme indiqué à la figure 20 : elle rend compte des obstacles et de la structure de grille.

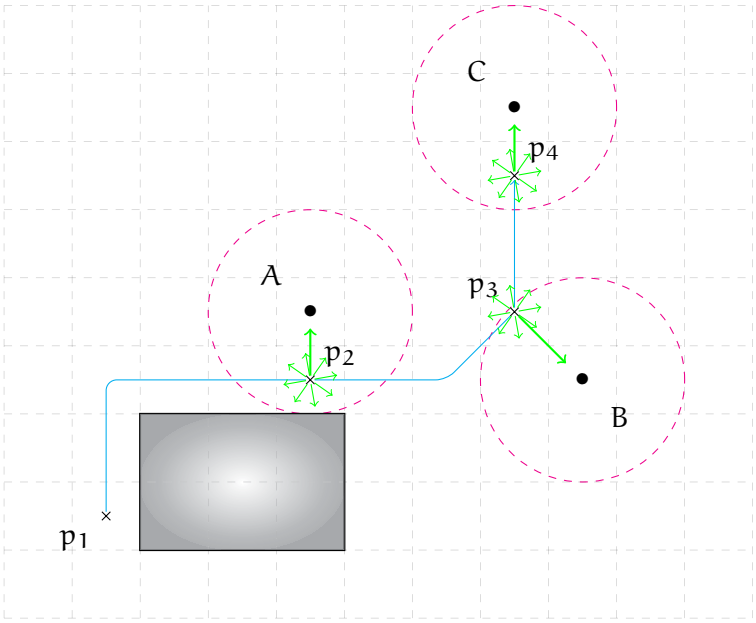


FIGURE 20 – Exemple de trajectoire planifiée à partir de modèles basés sur une grille. On reprend le contexte et les notations de la figure 18, mais ici les déplacements – et dans une certaine mesure la vision – sont contraints par l'obstacle et par la structure de grille. La nouvelle trajectoire parait moins optimale – elle est contrainte de passer par le centre des cases – et présente des virages serrés (angles droits) qui peuvent s'avérer difficiles à réaliser en pratique. Ce type de représentation basé sur une grille est très courant – voir la section 4.2.

On remarque que l'évitement d'obstacle ainsi que la structure de grille augmentent nettement la distance totale entre p_1 et p_4, en passant par p_2 et p_3. Elle présente aussi des virages serrés (angle droit) qui peuvent être problématiques pour un robot non holonome. En outre, la distance d'observation doit d'ailleurs elle aussi s'adapter à structure de grille. En particulier, suivant sa valeur, les observations « en diagonale » peuvent être ou non permises.

Supposons maintenant que robot n'est pas holonome et possède un champ de vue restreint à 120° seulement, ce qui contraint sa cinématique et rend son orientation très importante. Ces changements impactent fortement la forme de la solution, qui devient la trajectoire dessinée en bleu à la figure 21. En continuité des autres exemples, cela souligne combien la modélisation impacte le type de solution.

En fait, d'une manière plus générale, les modèles impactent aussi l'espace de recherche. Pour bien comprendre cela, reprenons les points solutions à

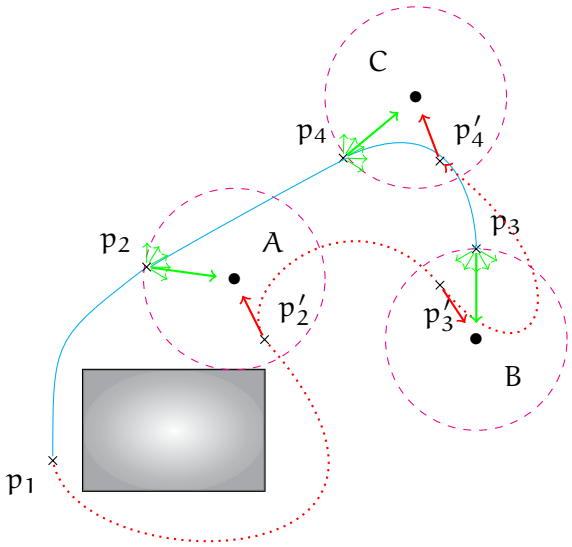


FIGURE 21 – Exemple de trajectoire planifiée à partir de modèles plus contraints, rendant compte de la cinématique du robot et de l'orientation de ses capteurs. On reprend le contexte et les notations des figures 18 et 20. Dans ce contexte, le plan d'origine $(p_1 \to p_2 \to p_3 \to p_4)$ ne semble plus tellement approprié, et on lui préférera largement le plan $(p_1 \to p'_2 \to p'_4 \to p'_3)$ comme alternative (en bleu).

la figure 19, et relions-les afin de former une trajectoire pour un robot non holonome, à champ de vue restreint. On obtient alors la trajectoire en pointillé de la figure 21, et l'on comprend bien que cette trajectoire $(p_1 \to p_2 \to p_3 \to p_4)$ n'a rien d'optimal. Cet écart s'explique en grande partie par le fait que les points (p_2, p_3, p_4) ont été choisis sans prendre en compte les contraintes cinématiques du robot. En d'autres termes, les modèles utilisés pour planifier n'étaient pas adéquats car ils s'appuyaient sur de mauvaises hypothèses (holonomie, champ de vision). Ainsi, il est important d'adapter l'espace de recherche aux modèles et aux capacités estimées des robots.

Ces trois exemples de modèles et la description de leur impact sur la planification illustrent deux points importants :

— La qualité des solutions est directement liée au réalisme des modèles utilisés lors de leur planification. Il s'entend ici que le terme « réalisme » désigne l'adéquation des modèles avec la réalité plutôt que leur précision : un modèle simple peut très bien être valide et réaliste s'il est conforme avec les capacités réelles des robots.

— En conséquence, ce sont les modèles qui doivent forger la solution ou du moins le type de solution, et par extension définir l'espace de recherche. Dans le cas contraire, on se retrouve dans l'incapacité de « boucler » entre le monde réel, les modèles le représentant, et le plan qui résulte de l'exploitation de ces modèles. En effet, l'écart

entre le résultat attendu et le résultat réel risque d'être trop grand, et il est même possible que les plans ne soient pas réalisables, par exemple en cas de trajectoire à la dynamique trop élevée. En résumé, les modèles ne permettent pas seulement une abstraction du monde réel : ils doivent également permettre de faire le chemin inverse et de « redescendre » d'une solution planifiée vers une suite d'actions valide.

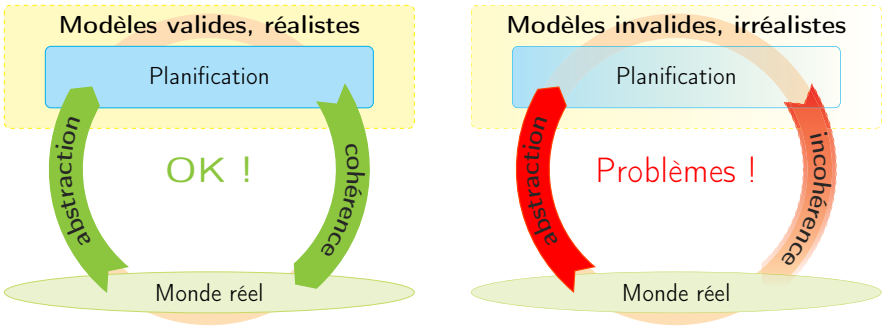


FIGURE 22 – Les modèles sont des passerelles entre le monde réel et sa représentation mathématique : il est important que celles-ci soient à double sens, c'est-à-dire de s'assurer que l'abstraction du monde reste réaliste et cohérente avec ce dernier, de manière à ce que les décisions soient valides dans les deux espaces, et non pas seulement dans la représentation mathématique.

Ces deux remarques ne signifient pas pour autant qu'il est impossible de travailler de manière abstraite : ainsi, dans le cas de la figure 18, les points $\{p_2, p_3, p_4\}$ ont été choisis en se basant sur un coût de déplacement défini par la distance euclidienne point à point. En considérant l'orientation du robot et en utilisant une autre fonction de coût (par exemple la distance de Dubins*), d'autres points auraient été choisis, pour un résultat plus proche de $\{p'_2, p'_3, p'_4\}$ – figure 21.

La distance de Dubins renvoie à la longueur de la plus petite courbe connectant optimalement deux points dans un espace euclidien à deux dimensions et respectant une contrainte sur la courbure maximale du chemin. Ce dernier est formé d'arcs de cercle reliés tangentiellement par des segments de droites.

Ainsi, les deux considérations mises en avant spécifient plutôt qu'il est important de ne négliger aucun des deux aspects de la modélisation : ni l'abstraction, c'est-à-dire le passage du monde réel à sa représentation mathématique, ni le réalisme ou la validité, c'est-à-dire le passage retour d'un plan mathématique à sa réalisation dans le monde réel – figure 22. Il est néanmoins important de garder à l'esprit que la difficulté du processus de la planification est directement liée aux hypothèses de simplification intégrées dans les modèles, ce qui pousse à rechercher le meilleur compromis entre simplification et réalisme.

Dans la suite de ce chapitre, nous développons plus en détails la relation entre modèles et planification (section 5.2), nous proposons une organisation particulière des modèles facilitant leur gestion (section 5.3) et nous décrivons l'intégration que nous en avons fait (section 5.4).

La planification s'appuie sur les différents modèles pour raisonner et prendre une décision quant aux actions à effectuer. Nous désirons cependant distinguer la notion de « modèles » (des agents et de l'environnement) de celle d'« état du monde », ou plutôt *des* états du monde.

Les modèles forment le champ sémantique permettant de décrire mathématiquement ou informatiquement le monde : ils permettent par exemple de représenter l'environnement sous forme de grille ou de graphe en décrivant le type de terrain (herbe, route, obstacle, *etc.*) de chaque case ou nœud. Ils peuvent aussi spécifier les capacités des capteurs et les capacités de mouvements des agents, ou encore indiquer les stratégies des cibles.

En l'absence de précision, l'« état du monde » renvoie à l'état estimé *du monde. L'expression « le monde » utilisée seule renvoie elle à l'état réel du monde.*

L'état du monde* est en planification la description mathématique effective de la situation (connaissances sur l'environnement, sur la position des protagonistes, *etc.*). En fait, il faut plus justement distinguer l'état réel du monde, qui est le monde tel qu'il est réellement, de l'état estimé du monde, qui est le monde tel qu'on le croit constitué, sa description issue des modèles. On regroupe dans la notion de monde l'environnement et les agents (robots, cibles, opérateurs).

Les modèles, autrement dit le champ sémantique, ne sont utiles que lorsqu'ils sont instanciés par l'état du monde (estimé), qui est la description effective du monde à l'aide de la sémantique des modèles. Par exemple, un modèle de mouvement peut spécifier « le robot roule aussi vite qu'il le peut, à une vitesse de 20km/h sur route et 5km/h ailleurs. » C'est alors l'état du monde qui précise, en donnant la position du robot et le type de terrain de la zone où il se situe, quelle est la vitesse attendue du robot.

Les modèles permettent ainsi de décrire le monde et son état, et en s'aidant de ce dernier, de prédire le résultat des actions entreprises par les agents. Mais dans le monde réel, la vitesse du robot sera peut-être légèrement différente de celle attendue et estimée, par exemple à cause d'un léger vent de face non modélisé pour un drone. Les modèles restent donc une approximation plus ou moins abstraite du monde réel, et il est important de garder cet écart à l'esprit.

Un plan est un ensemble d'actions partiellement ordonné.

La capacité de prédiction évoquée plus haut est essentielle à la planification. En effet, planifier consiste à élaborer une suite d'actions, appelée plan*, dans le but d'atteindre un état objectif prédéfini depuis un état initial connu ou estimé. Ce plan résulte d'un ensemble de simulations, estimant ou prédisant le résultat des différentes actions envisagées à partir d'un ensemble de modèles et de données connues du planificateur, dont l'état du monde.

Ces prédictions résultent de la « convolution » de modèles d'action avec des modèles de l'environnement, instanciés par l'état du monde – figure 23. Les modèles d'action sont donc dépendants des modèles d'environnement. En effet, les actions ne peuvent être entièrement modélisées de manière intrinsèque aux agents : leur faisabilité et leur résultat dépendent également

de l'état du monde au moment de l'action.

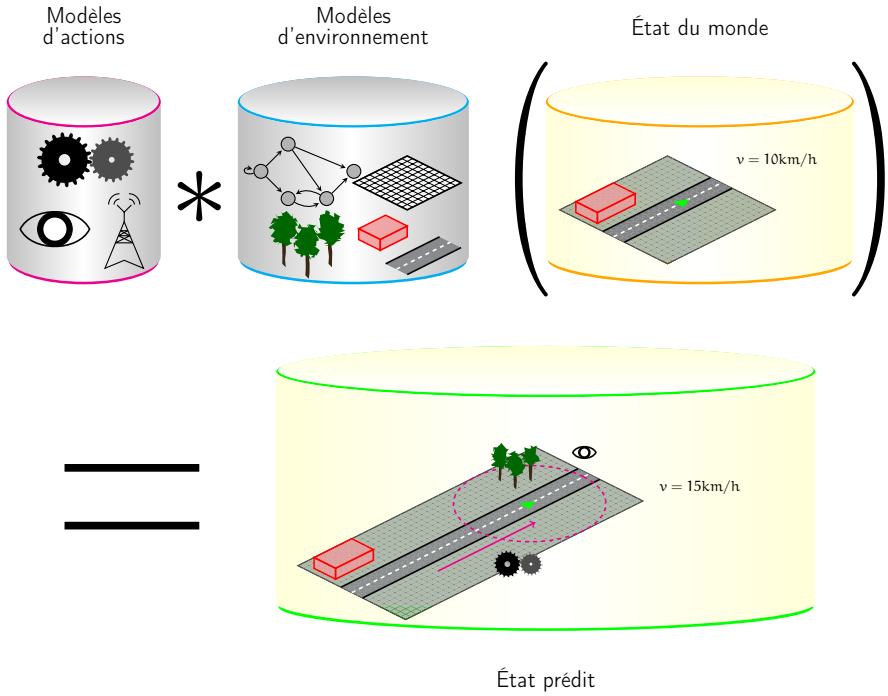


FIGURE 23 – La convolution des modèles d'actions et d'environnement, instanciés à l'état courant, décrit l'effet des actions envisagées et permet donc de prédire l'état résultant attendu. La simulation des effets actions (ici, une accélération couplée à une observation) fait partie intégrante du processus de planification.

Il est important de garder à l'esprit que ces prédictions ne sont utiles que si elles sont valides, c'est-à-dire cohérentes avec le monde. Elles n'ont pas nécessairement besoin d'être précises – elles peuvent être probabilistes, mais elles doivent rendre suffisamment compte de ce qui se passera effectivement dans le monde réel pour que le plan calculé reste valide et effectivement applicable en pratique, et non pas seulement dans l'abstraction mathématique.

Les agents considérés dans notre cas ont trois types d'actions :
— se déplacer,
— percevoir,
— communiquer.

Ces actions basiques peuvent se combiner pour former des « méta-actions » plus complexes, par exemple « percevoir en se déplaçant ». Chacune de ces actions peut en outre se décliner de plusieurs manières, entre différents robots ou au sein d'un même robot. Il existe ainsi plusieurs modes de déplacement (au sol ou dans les airs, à vitesse variable, *etc.*), de perception (Velodyne, LIDAR 2D, caméra panoramique, stéréovision, *etc.*) et de communication.

Nous décrivons plus en détails ci-dessous ces trois actions et les modèles associés :

SE DÉPLACER Planifier nécessite en robotique mobile de prédire et évaluer le déplacement des différents protagonistes. Le premier besoin concerné est la nécessité de trouver un chemin entre deux positions données, mais de manière plus large presque chaque action un peu élaborée nécessite de raisonner sur les déplacements. Dans notre contexte, ceux-ci sont en effet le principal moyen pour les robots d'« interagir » avec l'environnement. Pour autant, dans la famille de problèmes que nous considérons, se déplacer n'est que rarement une fin en soi, et s'avère plutôt être un moyen ou un sous-objectif induit : les robots se déplacent dans le but de communiquer ou de percevoir une autre partie de l'environnement que celle où ils se trouvent.

Les modèles liés aux déplacements incluent des cartes de traversabilité (décrivant différents types de terrain) et des modèles de mouvement (dynamique du robot, vitesses atteignables,, énergie consommée, *etc.*). Les modèles de déplacements incluent également les modèles de comportement, c'est-à-dire les stratégies de déplacement attendues pour les autres agents comme par exemple le comportement évasif d'une cible. Notons que l'action « se déplacer » est pour les agents considérés dans notre thèse la seule façon de modifier le monde.

PERCEVOIR Percevoir son environnement est une tâche essentielle pour les robots mobiles. C'est d'ailleurs en règle générale une tâche permanente et continue. Il y a néanmoins différents niveaux d'importance dans la perception suivant ce que le robot cherche à percevoir et son état propre (position, niveau de connaissance sur le monde). On peut ainsi distinguer différents sous-types d'action de perception, en fonction de leur objectif :

— percevoir pour augmenter les informations sur l'état du monde
— percevoir pour se localiser
— percevoir pour détecter/localiser/suivre une cible (« observer »).

Dans notre cas, c'est essentiellement cette dernière action qui nous intéresse et qui sera considérée dans la suite du document. Ainsi, bien que les autres actions de perception soient centrales pour interagir correctement avec le monde, nous les laisserons de côté : nous les considérons comme « passives », c'est-à-dire comme étant actives en permanence plutôt que sur requête, et nos plans ne spécifieront donc pas ces actions ni ne les prendront en compte pour, par exemple, influencer les déplacements.

Les modèles de perception sont bien souvent liés en premier lieu à la modélisation de l'environnement (hauteur et luminosité par exemple) et à des modèles spécifiques de capteurs, indiquant la portée de ceux-ci, leur angle de vision, leur sensibilité, *etc.*

COMMUNIQUER Les robots, lorsqu'ils sont déployés en équipe, ont souvent besoin de partager des données entre eux pour se coordonner. Un opérateur humain peut aussi demander un retour de la part des robots et

vouloir leur transmettre des ordres ou modifier la mission en cours. Communiquer devient alors un besoin fréquent. Afin d'intégrer des fenêtres de communication dans le plan envisagé, ou de trouver une bonne position pour permettre ces communications, les processus de planification ont besoin de modèles adéquats.

Ces trois types d'action et leurs modèles sont au cœur des processus décisionnels. Ils ont bien entendu des liens forts avec les modèles d'environnement : les types de terrain peuvent influencer le mouvement par exemple. En fait, chaque modèle d'action nécessite un modèle d'environnement rendant compte des propriétés considérées* : distances, obstacles, *etc.* Malheureusement, en pratique beaucoup d'auteurs ne distinguent pas les différentes actions, qui exploitent un seul et même modèle d'environnement à la sémantique limitée – voir section 4.2.1.

Les liens entre actions et environnements renvoient à la notion d'environnement multicouche – section 4.2.1.

L'état du monde instanciant les modèles contient les informations sur l'état des agents – comme la vitesse – et sur l'environnement – comme la position des obstacles. Comme expliqué précédemment, ces informations sont essentielles pour exploiter correctement les modèles d'action.

Par ailleurs, les modèles évoqués jusqu'ici sont souvent complétés par des utilités, qui sont des modèles spécifiques aux algorithmes de planification de la mission en cours – et non aux agents et à l'environnement bien qu'ils y soient nécessairement liés.

UTILITÉS Afin de choisir parmi différents plans possibles, le processus de planification associe souvent une valeur d'utilité aux différentes actions. Plus précisément, c'est généralement un ratio de cette utilité sur le coût de l'action envisagée qui permet d'évaluer chaque action, et la qualité des plans résultants.

Il existe de nombreux types d'utilités différentes : utilité de perception, de mouvement, de communication, *etc.* Elles sont propres à chaque problème de planification – elles valuent les objectifs et sous-objectifs – et sont habituellement calculées au besoin, en ligne. Leurs valeurs résultent en effet directement des effets des actions passées. Plus précisément, ces valeurs sont régies par des modèles d'évolution. On peut par exemple considérer l'utilité de percevoir une cible, et spécifier dans son modèle d'évolution que l'utilité sera proportionnelle au temps pendant lequel la cible est restée inobservée et inversement proportionnelle à la distance entre la cible et l'observateur.

Chaque planificateur peut définir ses propres utilités mais celles-ci restent fortement liées à d'autres données externes à la planification comme par exemple la précision courante de la localisation du robot. Malgré leur spécificité, il peut être intéressant de partager ces valeurs d'utilité avec les autres membres de l'équipe ou avec l'opérateur.

5.3 ORGANISATION DES MODÈLES ET DONNÉES ASSOCIÉES

Chaque robot intègre une base ou plusieurs bases de données constituées de données initiales, de modèles embarqués, de données acquises par communication ou issues de ses capteurs, ou encore calculées à partir d'autres données (localisation, cartographie). Ces bases de données sont centrales sur les plans des fonctionnalités et de l'autonomie. Elles sont en particulier cruciales pour les processus décisionnels comme décrit au début de ce chapitre. Chaque robot doit les organiser et les mettre à jour tout au long de sa mission. Nous proposons ici une façon de les organiser et de les gérer qui nous semble faciliter les problèmes associés aux modèles, comme la cohérence des données et le réalisme d'abstraction.

5.3.1 *Une gestion commune*

La section 5.2 a mis en évidence les différents modèles utilisés par la planification, souvent liés entre eux. Ces modèles sont définis au préalable et décrits de manière générique ; s'ajoutent ensuite des données initiales ou acquises en cours de mission, précisant ou complétant ces modèles.

Ces données associées évoluent au cours d'une mission : elles sont mises à jour suite à des actions de perception ou de communication (partage d'information entre les robots). Maintenir la cohérence des modèles et des données au cours du temps et entre les robots d'une même équipe est une tâche difficile à part entière.

Nous proposons pour cela de gérer les modèles et données – initiales et acquises – comme une seule et même base de données commune. En effet, il est courant que chaque module* intègre et gère lui-même ses propres données, à sa manière, pouvant être tour à tour client ou serveur de données. Nous pensons que cela complexifie le processus de maintien de la cohérence des données et des modèles, et explique en partie pourquoi peu de modules sont en pratique intégrés ensemble au sein d'une même plateforme embarquée : les modules sont plutôt testés indépendamment les uns des autres. Rassembler toutes les données en une seule et même base de données permet de gérer plus facilement leur cohérence.

Un module *est un ensemble de programmes considérés comme formant un tout, en charge d'une fonction spécifique (par exemple la localisation, ou la planification locale d'un chemin,* etc.*).*

Bien sûr, cela peut poser quelques difficultés – notamment sur la concurrence d'accès – mais celles-ci nous paraissent mineures par rapport au gain attendu. Pour fonctionner correctement, la gestion de ces données partagées doit être transparente à tous les modules concurrents y accédant, et être réalisée indépendamment par un module ou une librairie dédiée – voir figure 24.

Remarque. Cette base de donnée commune embarque autant les modèles et données liés à l'environnement que ceux liés aux agents et à leurs actions.

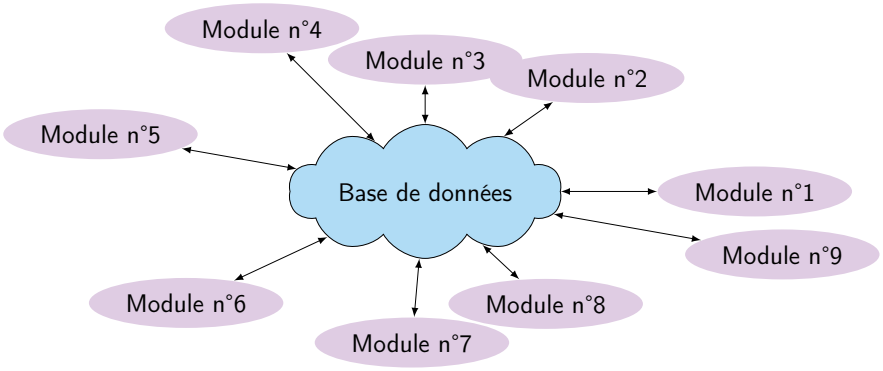


FIGURE 24 – Organisation globale des modules et données : chaque module interagit directement et indépendamment avec la base de données, à travers une interface commune. À charge de la base de données de maintenir sa cohérence interne à travers ses interactions. Les modules clients sont variés, allant de la perception à la planification, en passant par le SLAM et les modules de déplacements.

5.3.2 *Différents niveaux d'abstraction*

Outre la planification, les données et modèles embarqués sont aussi nécessaires pour la supervision, la cartographie et la localisation. Il est par ailleurs fréquent que plusieurs modules décisionnels fonctionnent en parallèle, ou en séquence, à des niveaux différents. Par exemple, la planification d'un chemin ou d'un déplacement local nécessite des données précises sur les obstacles environnants, alors que la planification d'un tour de garde à la manière d'un TSP ne nécessite que de savoir, pour deux positions données, s'il existe un chemin les reliant et le coût associé – il n'est pas nécessaire de savoir précisément quel est ce chemin.

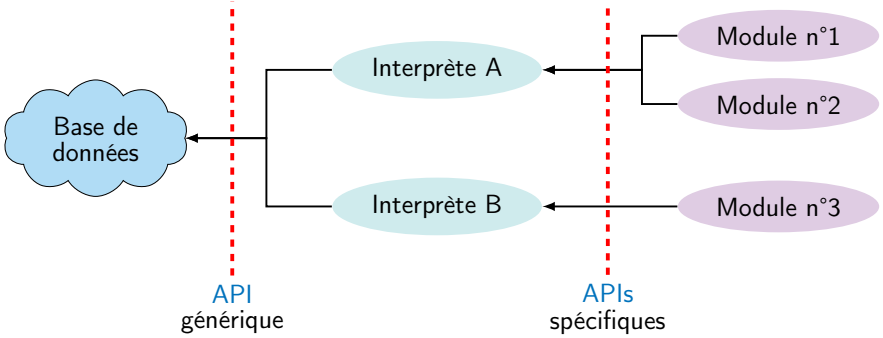


FIGURE 25 – Architecture à deux niveaux d'API pour la base de données des modèles : on utilise des modules intermédiaires comme « interprètes » pour traduire les données – généralement bas niveau – dans le formalisme attendu par les modules externes, et en particulier les planificateurs. Ainsi, les différents niveaux d'APIs correspondent à différents niveaux d'abstraction.

Les différents niveaux d'abstraction rencontrés travaillent pourtant sur la base des mêmes informations. Afin de mieux gérer ces différents niveaux d'abstraction, nous proposons d'utiliser différentes APIs , correspondant à

ces différents niveaux d'abstraction. Ainsi, au-delà de l'API donnant accès aux données brutes, nous rajoutons des modules « interprètes » s'interfaçant entre le client final et la base de données et donnant au client le niveau d'abstraction voulu à travers une API spécifique – voir figure 25. Ces modules interprètes peuvent être partagés entre plusieurs clients ayant des besoins similaires.

Ce type d'organisation hiérarchique utilisant des modules interprètes permet aussi de mieux maitriser le processus d'abstraction du monde par les modèles décrit à la section 5.2. Il permet de mieux s'assurer que chaque niveau d'abstraction est correct et permet d'élaborer des plans valides en retour.

Prenons l'exemple d'un module de planification résolvant un TSP : celui-ci n'a besoin, pour décider, que de connaître l'ensemble des points à visiter et le coût de déplacement point à point. Peu lui importe comment sont calculés ces coûts, ni où sont placés les points. Dans notre organisation des modèles, le niveau d'abstraction requis par le planificateur de TSP est fournit par un module planificateur plus bas niveau capable de calculer des trajectoires point à point et d'évaluer leur coût, par exemple à travers l'algorithme A* ou D*. C'est ce module interprète qui lit les données et exploite les modèles pour calculer des trajectoires valides et ainsi évaluer correctement les coûts.

Un autre avantage de cette organisation hiérarchique est qu'il est facile de substituer un module par un autre si les APIs sont respectées. Par exemple, on peut réutiliser notre planificateur de TSP pour des AAV comme pour des AGV simplement en changeant les modules de calculs. De même, un module d'exploration réutilisera facilement le module de calcul des coûts de déplacement point à point pour calculer des coûts d'accès aux frontières afin d'élaborer une stratégie d'exploration adéquate.

De manière plus concrète, imaginons que nous possédons un modèle de terrain numérique (MNT), et considérons deux robots de taille différente comme illustré à la figure 26. En convoluant le modèle d'environnement (MNT) et les modèles de déplacement des agents (incluant leur taille), on obtient deux graphes de déplacement distincts, un par agent, et fournissant la même API aux modules de planification – ils diffèrent par les chemins possibles.

Remarque. Notons que les modules interprètes nécessitent souvent des calculs intermédiaires permettant d'obtenir le niveau d'abstraction demandé. Comme mentionné dans les exemples, ceci peut inclure des processus de planification basiques comme des calculs de plus courts chemins. Ce sont ces calculs qui permettent de garantir la cohérence ascendante et descendante des différents niveaux d'abstraction entre eux.

5.3.3 *Géolocalisation*

Nous avons argumenté en faveur d'une base de données commune et partagée, permettant différents niveaux d'abstraction à travers des modules

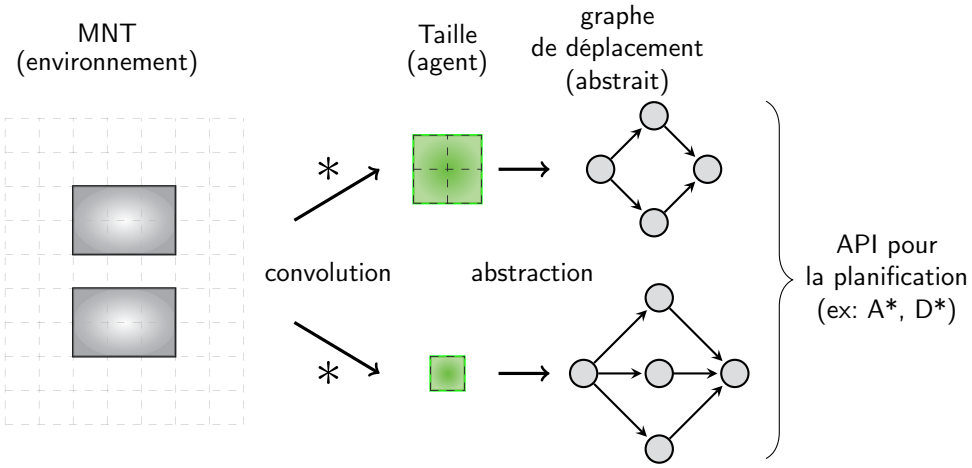


FIGURE 26 – Illustration du processus d'abstraction des modèles, appliqué au déplacement : en convoluant un même modèle numérique de terrain (MNT) avec deux modèles de robots différents, on obtient deux graphes différents mais proposant la même API d'accès aux données pour la planification.

interprètes. Dans la pratique, on observe cependant des besoins variés, notamment en matière de repères spatiaux. Ainsi, un module de localisation exploitant des données GNSS* a besoin d'un repère global – à l'échelle du monde – alors qu'un planificateur de chemin manipulera bien plus volontiers et facilement des coordonnées locales telles que « 100 mètres droit devant ». Pourtant, ces modules interagissent entre eux à travers la base de données commune.

GNSS est l'acronyme de Global Navigation Satellite System*, et fait référence à l'ensemble des systèmes de positionnement par satellite : GPS, GLONASS, Galileo, Beidou.*

Il est par ailleurs courant de posséder des données initiales issues de différentes sources, chacune utilisant un repère qui lui est propre. De même, les AGVs utilisent conventionnellement des repères *East-North-Up* (ENU) tandis que les AAVs utilisent plutôt des repères *North-East-Down* (NED) : une équipe de robots hétérogènes, souvent construits par différents laboratoires ou entreprises, doit pouvoir passer les données et objectifs de l'un des repères à l'autre de manière transparente. Il est donc nécessaire de trouver un langage commun permettant de faire le lien entre toutes ces données.

Pour cela, nous avons choisi de géolocaliser toutes nos données dans un repère monde* commun – en l'occurrence celui du GPS. En d'autres termes, chaque lot de données – une carte par exemple – utilisant un repère différent du repère commun fournit ou exploite un « traducteur » permettant de passer facilement des coordonnées d'un repère à l'autre.

Un repère « monde » situe les coordonnées à l'échelle de la planète. Un repère « local » situe les données par rapport à la situation courante, ou à un repère de référence locale comme par exemple la position de départ du robot.

Les bénéfices de ce choix sont multiples : chaque module peut travailler dans le repère de son choix tout en interagissant avec la base de données dans un repère commun à tous, ce qui garantit que toutes les données sont cohérentes. Il est alors facile d'y intégrer de nouvelles données, en fournissant simplement les fonctions de traduction si les repères différent. Les agents peuvent également s'échanger facilement des données, car elles sont par défaut dans le même repère.

Remarque. Il est important de noter que le géoréférencement des données ne réduit pas les incertitudes liées à cette localisation. Néanmoins, cet aspect là n'est pas abordé ici par la planification ni par les modèles développés et présentés ci-après : nous en discutons plutôt dans les perspectives de recherche. En conséquence, nous avons choisi dans les chapitres suivant de ne pas influencer les trajectoires ou les actions des robots pour réduire l'incertitude de localisation – par exemple en se déplaçant explicitement vers un repère connu et bien identifié.

Remarque. La synchronisation temporelle est aussi très importante, notamment pour déterminer l'obsolescence des données ou pour les regrouper. Elle est néanmoins beaucoup plus délicate lorsque l'on désire de la précision à cause des délais de transmission et de calcul au sein des plateformes embarquées, et des problèmes de synchronisation d'horloge. En pratique, nous effectuons l'horodatage des données aussi précisément que possible et nécessaire – un module de SLAM n'a pas besoin de la même précision qu'un module de planification de trajectoire. Nous négligeons ces aspects Au niveau de la planification, l'incertitude étant généralement bien en dessous des besoins du processus décisionnel.

5.4 INTÉGRATION

Dans les sections précédentes, et en s'appuyant sur notre analyse de l'état de l'art des problèmes de détection et pistage de cibles, nous avons défini et expliqué la notion de modèle et explicité de manière générale les besoins de la planification. Nous avons ensuite proposé une organisation spécifique des données et modèles au sein des robots. Celle-ci se base sur une séparation claire des modèles et des algorithmes d'une part, et des différents types de modèles d'autre part. En particulier, nous proposons une gestion commune des données aidée par un géoréférencement commun et une hiérarchie des modèles basée sur leur niveau d'abstraction.

Cela a pour but de mieux gérer la cohérence de ces derniers – instanciés par les données initiales et acquises ou calculées en cours de mission – en utilisant un module dédié à cette tâche particulière. Cela simplifie le travail des autres modules (planification, supervision, localisation), tout en facilitant voire en garantissant un retour valide au monde réel. En effet, les modules décisionnels n'ont pas à connaître en détails comment sont gérés l'ensemble des données et modèles : ils doivent simplement pouvoir accéder en lecture et en écriture aux données dont ils ont besoin, au niveau sémantique de leur choix. Pour cela, nous avons introduit la notion d'« interprètes », qui permettent de compléter l'API des modèles et données en ajoutant des précalculs si nécessaire, introduisant les niveaux d'abstraction requis.

L'organisation des modèles décrite à la section 5.3 a été intégrée au sein de nos robots et simulateurs sous la forme d'une librairie nommée *Gladys*, dont les détails techniques sont brièvement abordés à la section 5.4.1. La section 5.4.2 décrit les fonctions d'accès aux données implémentées dans

Gladys et utilisées dans la suite de cette thèse. Cette section se termine en décrivant ce qui est laissé à la charge des planificateurs, c'est-à-dire la notion d'état et les relations entre ces états – section 5.4.3.

5.4.1 *La librairie* Gladys

La librairie *Gladys*, conjointement développée avec Pierrick Koch, gère à la fois l'intégration de modèles et données ainsi que leurs cohérence et mises à jour. Elle fournit un ensemble de fonctions d'accès aux données, y compris à un niveau d'abstraction élevé tel que souhaité – voir section 5.4.2. La mise à disposition de modèles abstraits cohérents avec les données bas niveau doit permettre entre autres une intégration rapide d'algorithmes de planification évoluant sur des graphes ou exprimés dans un langage abstrait tel que PDDL. *Gladys* est actuellement utilisée avec succès par plusieurs modules de planification*. Outre plusieurs de nos travaux, nous pouvons notamment citer les travaux de Infantes et collab. exploitant *Gladys* comme fournisseur de modèles et de données pour leurs processus de planification [73].

Gladys est développée en C++ et Python et s'appuie sur les librairies GDAL pour le géoréférencement des données et Boost pour les structures de graphes. Les données d'entrées sont principalement des cartes fournies sous forme de fichiers .tiff contenant diverses méta-données, et des fichiers .json décrivant les robots (plateformes de locomotion, capteurs) et les missions (objectifs, positions de départs, données initiales connues, *etc.*). Des exemples de fichiers sont donnés en annexe A.

Le lecteur désirant approfondir les détails d'implémentation peut se reporter à l'annexe A décrivant plus en détails les fichiers utilisés et au code de la librairie* :

— `https://github.com/pierriko/gladys`
— `http://trac.laas.fr/git/gladys`

Gladys *est distribuée sous license BSD-3.*

5.4.2 *Fonctions d'accès aux données*

Le tableau 1 décrit les fonctions actuellement développées dans *Gladys* ou en développement. Ce sont ces fonctions qui sont utilisées par la planification pour interroger les modèles et données afin de pouvoir décider à propos. Nous avons regroupé les fonctions d'accès aux données selon les trois grands types d'action : le mouvement, la perception et les communications.

Nous ajoutons également des requêtes liées à l'utilité : contrairement aux fonctions précédentes, celles-ci dépendent du planificateur et de la mission : chaque planificateur les instancie différemment. Nous les regroupons néanmoins ici par soucis de clarté et de lisibilité.

Les notations utilisées dans cette section, notamment pour les valeurs retours, sont reprises par la suite aux chapitres 6 et 9 ; elles sont égale-

Fonction	Type	Arguments	Valeurs retour	Algorithmes
is_accessible	MOUVEMENT	a, p, c_{max}	$\{ p' \}$	Dijksrta
simulate	MOUVEMENT	a, p, c_{max}	$\{ (p', \pi^a_{p'}) \}$	Proba, MDP
navigate	MOUVEMENT	r, p_1, p_2	$path$, $c^r_{p_1p_2}$	A*, D*
test_visibility	PERCEPTION	r, p, q	ϕ^r_{pq}	†
can_see	PERCEPTION	r, p	$\{ (q, \phi^r_{pq}) \}$	†
is_visible_from	PERCEPTION	r, q	$\{ (p, \phi^r_{pq}) \}$	†
test_comlink	COMMUNICATION	r, p_1, p_2	$\alpha^r_{p_1p_2}$	‡
can_broadcast	COMMUNICATION	r, p	$\{ (p', \alpha^r_{pp'}) \}$	‡
is_observed	UTILITÉ	q	u	–
is_reached	UTILITÉ	r, p	u	–
is_travelled	UTILITÉ	$r, path$	u	–

TABLE 1 – Les différentes fonctions d'accès aux données implémentées dans *Gladys*. Une partie d'entre elles est directement exploitée par les algorithmes présentés aux chapitres suivants

ment rassemblées dans la nomenclature (« liste de symboles »). Nous les détaillons ci-dessous, et précisons au passage les fonctions associées :

NOTATIONS GÉNÉRALES

- $r \in R$ désigne un des robots disponibles pour la mission en cours.
- $a \in A$ désigne un des agents (robots, cibles, opérateurs) protagoniste de la mission en cours.
- Q est la zone observable ; $q \in Q$ désigne une position observable. Dans la plupart des missions, la zone observable est également la zone *à observer* (par exemple en mission de patrouille ; voir le chapitre 6).

- P^a est la zone accessible à l'agent $a \in A$; $p \in P^a$ est une des positions accessibles, par exemple celle où se situe l'agent a. Dans le cas spécifique des robots, on pourra utiliser la notation P^r. Il n'est pas fait d'hypothèse sur le nombre de dimensions des P^a : ceux-ci peuvent donc notamment inclure l'orientation des agents.

MODÈLES DES ACTIONS

- MOUVEMENT Pour un robot r donné et $(p_1, p_2) \in P^r \times P^r$, nous notons $c^r_{p_1p_2}$ le coût de déplacement de p_1 à p_2 pour r. Suivant les spécifications de la mission, ce coût peut être exprimé dans des unités différentes : distance, temps, énergie, *etc.*.
  - ☐ is_accessible indique quelles sont les positions $\{p'\}$ accessibles à l'agent a – robot ou cible – depuis la position p pour un coût inférieur à c_{max}.
  - ☐ simulate est très similaire : la différence tient dans la prise en compte d'un modèle probabiliste de comportement pour l'agent

a considéré – pour la cible par exemple – permettant d'évaluer la probabilité de présence $\pi^a_{p'}$ de l'agent à chaque position p'.

- □ navigate donne explicitement un chemin entre deux positions p_1 et p_2 pour un robot donné. La fonction sert aussi à calculer le coût minimal c^r de navigation entre ces deux points.

— Perception Étant donnés $q \in Q$ et $p \in P^r$, $\phi^r_{pq} \in [0,1]$ indique la qualité de perception de q par r^* depuis p, 1 correspondant à une perception parfaite et 0 à une absence de perception. Chaque capteur possède sa propre fonction ϕ rendant compte de toutes ses propriétés, et notamment sa portée et son angle de vue.

Nous ne considérons pas dans nos travaux le point de vue de la cible ; c'est pourquoi la perception n'est définie ici que pour les robots.

- □ test_visibility indique la qualité de perception ϕ^r_{pq} d'une position q depuis une position p, pour un robot r donné et tous capteurs confondus.
- □ can_see indique toutes les positions q observables par le robot r depuis la position p ; elle spécifie aussi la qualité de perception.
- □ is_visible_from est sa réciproque : elle indique depuis quelles positions p la position q est observable et avec quelle qualité ϕ^r_{pq}.
- † Les fonctions de perception calculent la visibilité à partir d'un « lancer de rayons » pour les obstacles à la vision et de la distance entre les positions pour la qualité de perception. Il est possible d'ajouter un masque affectant la qualité de perception, modélisant par exemple la luminosité ambiante.

— Communication Nous modélisons les communications comme suit : considérant deux robots r_1 et r_2 situés respectivement en p_1 et p_2, ils peuvent communiquer l'un avec l'autre ssi :

$$\alpha^{r_1}(p_1, p_2)\alpha^{r_2}(p_2, p_1) = 1 \tag{1}$$

avec $\alpha^r(p_1, p_2) : P \times P \to \{0,1\}$ une fonction binaire indiquant si le robot r peut communiquer ($\alpha = 1$) ou non ($\alpha = 0$) avec la position p_2 depuis p_1.

- □ test_comlink indique la qualité $\alpha^r_{p_1 p_2}$ du signal de communication entre deux positions p_1 et p_2 pour un robot r donné.
- □ can_broadcast indique toutes les positions capables de recevoir un signal émis par r depuis p ; la fonction précise là encore la qualité de la liaison.
- ‡ Les fonctions de communications calculent la qualité de la liaison à partir d'un « lancer de rayons » pour les obstacles à la communication et de la distance entre les positions pour la puissance du signal.

coût et utilité

— Coût Actuellement, seules les actions de déplacements sont associées à un coût, dont l'unité est généralement spécifiée par l'objectif ou les contraintes de la mission.

— Utilité Actuellement, seules les actions de perception sont associées à des utilités. Néanmoins, si l'on considère une perception continue (ce qui est le cas en pratique), il est également possible d'associer une utilité à une action de déplacement. En revanche, aucune utilité n'est associée à la communication, qui est considérée comme une contrainte dans nos travaux. Nous utilisons la notation $\mathfrak{u} : Q \times \mathbb{R}^+ \to \mathbb{R}^+$ pour rendre compte de l'utilité de perception d'une position $q \in Q$. En pratique, l'utilité $\mathfrak{u}$ est souvent pondérée par la qualité de perception ϕ^r.
  - ☐ $\mathrm{is\_observed}$ indique l'utilité d'observer la position q, lorsqu'une telle utilité est définie. Comme aucun modèle de perception n'est pris en compte par cette fonction, la valeur d'utilité renvoyée correspond par défaut à celle obtenue pour une observation parfaite.
  - ☐ $\mathrm{is\_reached}$ indique l'utilité d'atteindre la position p pour le robot r. Cela sous-entend qu'une action de perception est effectuée en p : la qualité de perception est prise en compte.
  - ☐ $\mathrm{is\_travelled}$ indique l'utilité de parcourir le chemin path pour le robot r. Tout comme pour $\mathrm{is\_reached}$, cela sous-entend que des actions de perception sont effectuées tout au long du chemin ; la qualité de perception est prise en compte.

Les fonctions regroupées dans le tableaux 1 sont le résultat de l'analyse de nos besoins et des principes mis en avant tout au long de ce chapitre. Elles exploitent les modèles évoqués plus haut et identifiés dans la première partie de nos travaux, aux sections 4.2.2 « modèles des agents » et 4.2.1 « modèles d'environnement ». Ces fonctions permettent l'accès aux données pour un état donné, et sont donc utilement complétées par des fonctions permettant de gérer cette notion d'état. Ceci est développé à la section suivante. L'ensemble est exploité par différents planificateurs dans les chapitres suivants.

5.4.3 *Gestion des états*

Nous nous intéressons ici à la gestion des états estimés du monde. Comme expliqué précédemment, un état estimé est pour nous un ensemble de données initiales, acquises ou calculées exploitant – instanciant – des modèles et permettant de rendre compte d'un état réel du monde, actuel, passé ou hypothétique.

Il existe donc *des* états estimés du monde. En effet, il peut être utile pour le planificateur de mémoriser divers états estimés hypothétiques afin de limiter les calculs redondants et de permettre une replanification rapide. Il est également intéressant de garder une trace des actions et des états

du monde passés. Cela peut s'intégrer à un processus d'apprentissage ou à l'évaluation des utilités ou bien à des processus externes à la planification tels que la supervision ou la fermeture de boucles en localisation par exemple. Mais il est aussi possible de simplement se rappeler que telle situation est un état connu qui s'est déjà produit.

Si la base de données commune contient les données associées à ces divers états, nous laissons à chaque planificateur le soin de gérer les liens entre ces états. Les états sont généralement rassemblés dans un graphe d'états, décrivant diverses situations (= les états) et comment l'on passe de l'une à l'autre à travers les actions des agents, comme illustré à la figure 27.

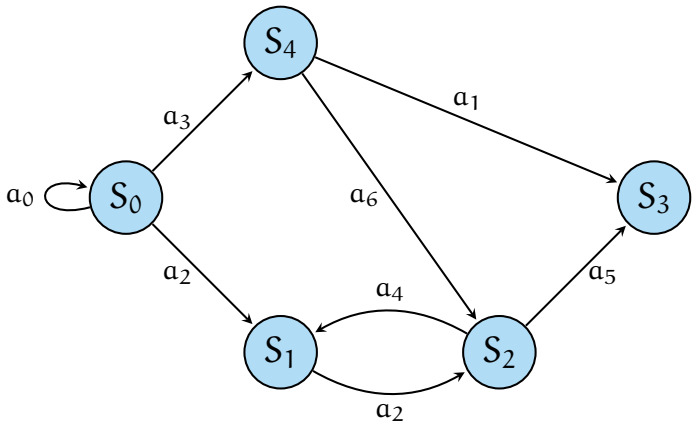


FIGURE 27 – Exemple de graphe d'états : chaque nœud S_i correspond à un état spécifique ; les transitions entre les nœuds sont les actions a_j transformant l'un en l'autre.

Tous les planificateurs ne modélisent pas explicitement la notion d'état – voir le chapitre 6 par exemple – mais lorsqu'ils le font, les fonctions suivantes sont utiles voire nécessaires – voir chapitre 9 :

— get_current_state( ) : charge en mémoire les données de l'état estimé courant ;
— create_new_state(action) : créé un nouvel état en appliquant action à l'état actuellement considéré ;
— load_state(ID) : charge en mémoire les données de l'état estimé ID ;
— delete_state(ID) : efface l'état ID de la mémoire ;
— clear_states( ) : efface la mémoire de tous les états passés ou hypothétiques, mais garde les données associées à l'état courant.

Les fonctions décrites à la section 5.4.2 permettent un accès aux données pour un état en particulier désigné avec la fonction load_state et choisi par le planificateur. Ainsi, les fonctions sont accessibles pour chaque état : elles permettent au planificateur d'accéder aux informations guidant sa prise de décision, c'est-à-dire à la sélection d'une action ou d'une suite d'actions définissant alors un nouvel état.

En résumé, nous avons exhibé dans ce chapitre l'importance des modèles et leur impact sur les processus décisionnels. Nous avons proposé une définition claire des notions associées et une certaine organisation permettant de mieux gérer les liens entre le monde réel et les données d'une part, et la planification d'autre part. Les fonctions décrites à la fin de ce chapitre instancient et mettent en pratique ces idées. Elles implémentent la séparation claire des modèles et des algorithmes, ce qui nous permet de faire évoluer facilement les uns et les autres de manière indépendante tout en encourageant l'utilisation de modèles plus riches dans les processus de décision. Les chapitres suivants exploitent ces fonctions et ces idées pour développer différents algorithmes de planification, et étudient l'impact de modèles plus riches sur les performances des algorithmes dans le cas particulier de la patrouille et du suivi de cible.

6

PATROUILLE DE ZONES SÉCURISÉES : FORMALISME ET THÉORIE

Tout l'art de la guerre est basé sur la duperie.

— Sun Tzu

Dans ce chapitre, nous abordons le problème de patrouille appliqué au cas d'une zone sécurisée. Ce type de mission consiste en une tâche cyclique de surveillance d'une zone donnée et connue, avec la spécificité d'un contexte antagoniste – voir aussi la section 2.4 pour plus de détails. Dans ce contexte, nous proposons plusieurs schémas de patrouille basés sur différentes formulations et résolutions du problème. La première partie de ce chapitre décrit les notations et les modèles utilisés, tandis que la seconde partie aborde la résolution théorique du problème. Les résultats sont présentés et analysés dans les chapitres suivants.

6.1 CONTEXTE ET APPROCHE

Nous pouvons résumer l'état de l'art relatif aux problèmes de patrouille présenté précédemment à la section 2.4 par les principaux constats suivants :

- La zone à patrouiller est souvent modélisée à l'aide de graphes.
- Le problème est NP-complet dans sa forme générale.
- Le principal critère d'évaluation d'un schéma de patrouille, appelé l'*oisiveté*, correspond au temps écoulé entre deux visites d'une même position.
- Les cycles hamiltoniens forment les schémas de patrouille optimaux – lorsqu'ils existent et à priorités égales entre les nœuds du graphe.
- La principale alternative consiste à diviser le graphe en sous-graphes, en particulier lorsque celui-ci n'est pas hamiltonien ou contient des nœuds de priorités différentes.
- Les schémas optimaux sont déterministes donc prédictibles.
- Il existe des schémas non prédictibles mais ils ne donnent aucune garantie sur leurs performances ou la qualité des solutions.

S'il peut être intéressant de patrouiller optimalement une zone, l'aspect déterministe des solutions optimales n'est pas adéquat en contexte antagoniste ; il s'avère même contre-productif. Il est alors nécessaire de se pencher sur des solutions non déterministes, sous-optimales. Cela se fait généralement en introduisant un aspect aléatoire dans le processus de décision. Nous cherchons ici à élaborer de tels schémas de patrouille, stochastiques et non prédictibles, mais en fournissant autant que possible des garanties sur leurs performances : l'aspect imprédictible des solutions ne doit pas se

faire au prix de la qualité de celles-ci. Nous cherchons donc des solutions imprédictibles « presque optimales ».

L'idée principale derrière notre approche est la notion d'*instance* : nous appelons instance d'un problème de patrouille un problème « tronqué » tiré du problème d'origine, obtenu par un échantillonnage stochastique du problème d'origine pour une période de temps bornée – notée T dans la suite du document. On cherche alors à résoudre optimalement ce nouveau problème, définissant ainsi un schéma de patrouille jusqu'au prochain échantillonnage. Ce processus constitué de l'étape d'instanciation et de sa résolution est donc cyclique. L'échantillonnage agit comme un curseur entre l'optimalité, la (non-)prédictibilité et la complexité – voir la section 6.3. Lorsque l'échantillonnage est exhaustif, on se retrouve avec le problème d'origine. Plus l'échantillonnage est faible et aléatoire, et moins le problème, et donc sa solution, sont prédictibles. La taille de l'échantillonnage permet enfin de gérer la complexité du problème.

Un autre argument en faveur d'une méthode par échantillonnage est la notion d'« élasticité » des solutions optimales, mise en avant par Gonzales et Latombe [65] – voir p. 14. Pour rappel, celle-ci stipule que les solutions optimales souhaitées peuvent être modifiées légèrement sans changement notable de la qualité de la solution, ce qui est une preuve de robustesse désirable en pratique.

La résolution optimale du problème instancié peut se faire de bien des manières. Comme expliqué dans les sections suivantes, nous représentons le problème instancié par un graphe ; néanmoins, les cycles hamiltoniens ne forment pas une solution adaptée ici car, comme nous allons le voir, l'existence de tels cycles n'est pas garantie, et la considération du critère d'oisiveté fait que les nœuds ont des priorités différentes. Nous proposons donc deux approches alternatives : un algorithme de type Dijkstra – section 6.4 – et une résolution par programmation par contraintes, et plus précisément par optimisation en nombres entiers – section 6.5.

Les algorithmes présentés respectent par ailleurs nos recommandations du chapitre 5 et permettent l'usage de modèles réalistes décrivant l'environnement et les agents. Cet aspect affecte autant l'étape d'échantillonnage que la résolution du problème instancié, et implique autant les modèles de perception et de communication que ceux des déplacements. Ces derniers sont laissés sous forme de points de passage auxquels est associée une fonction d'estimation du coût de déplacements, conformément à notre hiérarchisation des modèles et des niveaux d'abstraction ; les détails des trajectoires sont alors laissés à d'autres modules de planification et à un contrôle bas niveau entre les points de passage.

Dans les sections suivantes, nous décrivons le formalisme et les modèles utilisés – section 6.2 – et discutons de l'échantillonnage comme compromis entre la prédictibilité et l'optimalité – section 6.3. Nous proposons ensuite

de résoudre optimalement le problème instancié directement sur le graphe à l'aide d'un algorithme inspiré de l'algorithme de plus court chemin de Dijkstra – section 6.4– et fournissons différentes formulations permettant de résoudre le problème instancié comme un problème de programmation par contraintes – section 6.5. Nous terminons ce chapitre par une discussion de notre approche – section 6.6 – en abordant notamment l'aspect cyclique du problème et l'impact des communications sur sa complexité.

6.2 Modélisation du problème

Nous supposons que la zone à patrouiller est connue. En particulier, nous connaissons les zones accessibles et la présence d'obstacles, les zones observables et la qualité d'observation, ainsi que les positions permettant d'établir des liens de communications entre elles. Nous supposons de même que les modèles suivants sont disponibles pour chacun des agents patrouilleurs : perception, déplacements et communications. Nous reprenons les notations de la section 5.4.2 ; elles sont complétées par les notations listées ci-dessous. Nous détaillons également certaines hypothèses sur les modèles mathématiques.

Notations générales (rappel)

- R désigne l'ensemble des robots r disponibles pour la mission de patrouille. On note $N = \mathrm{card}(R)$ le nombre de robots.
- Q est la zone à patrouiller ; nous utilisons la notation $q_k \in Q$ lorsque l'on considère une version discrète du problème – par exemple après échantillonnage – et $m = \mathrm{card}(Q)$ le nombre de positions objectifs.

- P^r est la zone accessible à l'agent $r \in R$; nous utilisons la notation $p_i^r \in P^r$ dans la version discrète du problème.

Modèles des actions

- Perception Sans que cela ne restreigne réellement le problème, nous considérons que chaque robot r a un unique capteur* lui permettant de surveiller ses alentours à la recherche d'intrus. Pour rappel, étant donnés $q \in Q$ et $p^r \in P^r$, nous avons $\phi^r(p^r, q) \in [0, 1]$ indiquant la qualité de perception de q par r depuis p^r. Pour simplifier les notations, nous utilisons $\phi^r_{ik} = \phi^r(p^r_i, q_k)$ dans le cas discret.

- Mouvement L'ensemble des $p_i^r \in P^r$ constituent un graphe d'accessibilité $\mathcal{G}^r(\mathcal{A}^r, \mathcal{V}^r)$ reliant les positions entre elles et décrivant les déplacements possibles pour le robot r. Nous utilisons le formalisme usuel des graphes : $\mathcal{A}^r$ est l'ensemble des arcs (i, j) et $\mathcal{V}^r$ l'ensemble des sommets. On note n^r le nombre de sommet de V^r – qui est donc aussi le cardinal de P^r – et on a $a^r_{ij} = 1$ si et seulement si le robot

Lorsque le robot possède plusieurs capteurs externes, le capteur considéré ici est un capteur virtuel dont les capacités rendent compte de tous les capteurs du robots réunis, comme si l'on choisissait le plus adapté (ou la combinaison la plus adaptée) à chaque situation.

peut se déplacer du sommet i associé à p_i^r au sommet j associé à p_j^r, $a_{ij}^r = 0$ sinon. Le coût de déplacement de p_i^r à p_j^r est noté c_{ij}^r ; conformément à notre API hiérarchique présentée section 5.3.2, nous supposons qu'il est évaluable à la demande, à l'aide d'une fonction ou d'un module approprié. Nous introduisons aussi P, le produit des P^r. De même, $\mathcal{G}(\mathcal{A}, \mathcal{V})$ est le produit des $\mathcal{G}^r(\mathcal{A}^r, \mathcal{V}^r)$.

— COMMUNICATION Nous reprenons les mêmes notations et hypothèses que dans la section 5.4.2.

COÛT ET UTILITÉ

— COÛT Nous supposons les coûts des actions – des déplacements notamment – valués en temps. Nous instancions et planifions alors pour un horizon temporel fini T, qui peut être interprété comme le coût maximal d'un plan par robot.

— UTILITÉ Nous considérons également un modèle d'utilité $u : Q \times \mathbb{R}^+ \to \mathbb{R}^+$ qui rend compte tout autant de l'*oisiveté* – fonction du temps $t \in \mathbb{R}^+$ – et de l'importance de la position considérée – fonction de $q \in Q$. Soient $q \in Q$ et t le temps, l'utilité $u(q, .)$ est une fonction croissante de t tant que q n'est pas observée. De plus, $u(., t)$ est bornée sur Q à t fixé. Lorsqu'un robot r observe q depuis p^r, l'utilité décroît, éventuellement jusqu'à zéro, suivant l'équation :

$$u^+(q, t) = u^-(q, t) * (1 - \phi^r(p^r, q)) \tag{2}$$

u^- et u^+ étant les utilités de la position q respectivement avant et après l'observation – rappelons que $\phi^r(p^r, q) \in [0, 1]$.

Remarque. Des exemples de ces modèles et fonctions (ϕ^r,u, c_{ij}^r,α^r, *etc.*) sont donnés à la section 7.1, avant de montrer les résultats associés.

6.3 ÉCHANTILLONNAGE ET INSTANCIATION

Avant échantillonnage, les modèles peuvent être indifféremment continus ou discrets. Nous souhaitons ici les échantillonner pour obtenir des données adéquates pour nos algorithmes, c'est-à-dire pour obtenir un problème instancié. L'échantillonnage ne doit pas être régulier – comme on le ferait en suivant une grille – mais au contraire stochastique de manière à créer des instances « inattendues ».

Le problème est essentiellement défini par la zone à observer Q et l'espace des solutions par les positions accessibles P^r. La zone à observer est échantillonnée par rapport à l'utilité pour chaque position $q \in Q$ d'être ob-

servée. L'utilité est donc transformée en η, une fonction de pseudo densité de probabilité, ou *Probability Density Function* (PDF) :

$$\eta : Q \to [0, 1] \text{ avec } \eta(q) = f(u(q)) \tag{3}$$

avec $f : \mathbb{R}^+ \to [0, 1]$ une fonction choisie et u la fonction d'utilité telle que définie à la section 6.2. Le choix de f est donc crucial* pour le compromis entre la prédictibilité et l'optimalité car elle peut soit accentuer soit lisser les variations de u sur Q.

Pour mieux saisir l'importance de f, le lecteur est invité à comparer le choix d'une fonction quadratique et d'une fonction racine carrée.

Un autre paramètre essentiel de l'échantillonnage du problème est le nombre d'échantillons, car il définit la proximité du problème avec celui d'origine tout en déterminant la taille du problème instancié. Nous proposons trois méthodes d'échantillonnage de P^r, détaillées dans les sous-sections suivantes ; chacune engendrant une formulation différente du problème.

6.3.1 *Échantillonnage orienté positions*

Dans cette formulation du problème, les modèles de capteurs sont simplifiés de la manière suivante :

$$P^r \subset Q \tag{4}$$

$$\phi^r(p^r, q) = \begin{cases} 1 & \text{si } q = p^r \\ 0 & \text{sinon} \end{cases} \tag{5}$$

En d'autres termes, un robot ne peut observer que la position à laquelle il se trouve – voir la figure 28 – mais celle-ci est parfaitement observée. Dans cette configuration particulière, on peut considérer que P^r et Q sont en quelque sorte « fusionnés ».

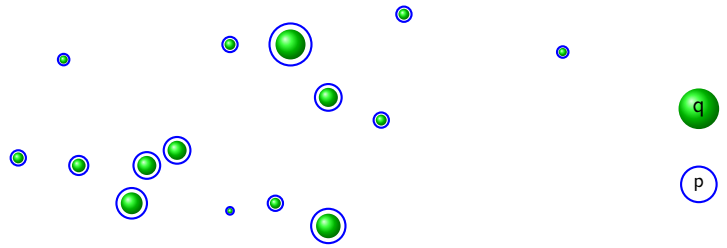


FIGURE 28 – Échantillonnage avec un modèle basique de capteur dans lequel les robots observent parfaitement la position où ils se trouvent, mais pas les alentours. Les positions accessibles (cercles bleus) et observables (boules vertes) sont confondues. La taille des boules et des cercles est proportionnelle à l'utilité associée. Ce modèle de capteur, bien qu'extrêmement basique, est très répandu dans la littérature – cf. chapitre 5.

De cette manière, en confondant P^r et Q, on échantillonne uniquement les $\{p_i^r\ ,\ i \in [1, n^r]\} \subset P^r$ pour chaque $r \in R$ afin de construire le graphe $\mathcal{G}^r(\mathcal{A}^r, \mathcal{V}^r)$, en utilisant la PDF

$$\eta^r(p^r) = f(u_{|_{P^r}}(p^r)) \tag{6}$$

avec $u_{|_{P^r}}$ la restriction de u à P^r. Le produit de ces graphes définit l'espace de recherche des solutions. Le problème est lui-même défini par les $q \in Q$ identiques aux positions accessibles échantillonnées : il ne sert en effet à rien d'échantillonner d'autres positions observables car d'après le modèle de capteurs considéré ici, celles-ci ne pourront être prises en compte lors de la planification.

6.3.2 *Échantillonnage orienté perceptions*

Dans cette seconde façon d'instancier le problème, ϕ^r n'est pas restreint et suit sa définition initiale $\phi^r : Q \times P^r \to [0, 1]$. Les P^r ne sont pas contraints par Q. Ainsi, de même qu'à la section précédente, il n'est pas nécessaire d'échantillonner Q et l'échantillonnage se fait sur les $\{p_i^r\ ,\ i \in [1, n^r]\} \subset P^r$ pour chaque $r \in R$ afin de construire le graphe $\mathcal{G}^r(\mathcal{A}^r, \mathcal{V}^r)$. Les $q \in Q$ pris en compte sont ceux reliés par les modèles d'observation aux p^r échantillonnés, les autres sont ignorés par la planification. Nous utilisons la PDF suivante pour définir l'espace des solutions :

$$\eta^r(p^r) = f(\nu^r(p^r)) \tag{7}$$

avec la « fonction d'utilité (cumulée) »

$$\nu^r : \begin{cases} P^r \to \mathbb{R}^+ \\ \nu^r(p^r) = \int_Q \phi^r(p^r, q) u(q) dq \end{cases} \tag{8}$$

$\nu^r(p^r)$ peut être interprété comme la récompense d'un robot r atteignant p^r et observant ses alentours dans Q, « récoltant » l'utilité des zones perçues – voir la figure 29.

Remarque. (8) nécessite implicitement que $\int_Q \phi(p^r, q) u(q) dq$ soit borné, car une valeur infinie de $\nu^r(p^r)$ n'est pas souhaitable. Cette hypothèse est généralement vérifiée puisqu'on a la contrainte $\phi \in [0, 1]$: en supposant u bornée au moment de l'échantillonnage – c'est-à-dire sans valeur infinie – et en supposant Q borné également (ce qui correspond à définir un espace fini pour la mission : c'est toujours vrai en pratique !), alors on a bien les conditions pour que ν^r n'ait que des valeurs finies.

6.3.3 *Échantillonnage multiple perceptions-positions*

Dans cette troisième façon d'instancier le problème, nous décorrélons totalement les P^r et Q : ils sont tous échantillonnés indépendamment –

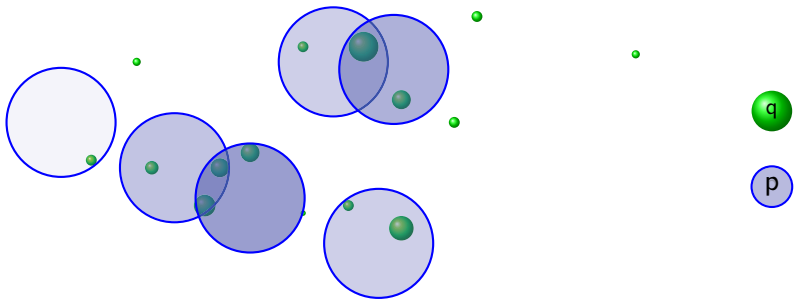


FIGURE 29 – Échantillonnage avec un modèle grossier de capteur dans lequel un robot observe les positions proches de sa position, mais ses différentes observations sont considérées comme indépendantes. Cela peut mener à des observations multiples d'une même zone sans que cela soit détecté par la planification. L'utilité des positions accessibles est indiquée par leur opacité. Ce modèle est plus avancé que le modèle basique de capteur, mais il est mal exploité car il prend en compte de multiple fois l'utilité d'une même position observable : il favorise ainsi les observations multiples d'une même position, ce qui n'est pas désiré dans notre cas.

voir la figure 30. La définition de l'espace de recherche des solutions – les $p^r \in P^r$ – et l'instanciation du problème – les $q \in Q$– sont donc clairement séparés, et ce sera la phase de résolution qui fera les liens entre eux. L'instance ne comporte donc pas de choix *a priori*, contrairement aux échantillonnages précédents. L'échantillonnage de l'espace de recherche des solutions constitué par les P^r n'est pas contraint par Q : on est donc libre de l'échantillonner comme désiré. Q est lui échantillonné selon son utilité à l'aide de (3). Nous désignons ici par n^r le nombre d'échantillons de P^r et m celui de Q.

Remarque. Afin d'explorer un espace de recherche aussi vaste et régulier que possible, nous choisissons par la suite d'échantillonner les P^r de manière régulière à l'aide d'une PDF constante ν^r, comme illustré à la figure 30.

ϕ^r suit sa définition initiale. Contrairement aux formulations précédentes, la perception n'est pas prise en compte lors de l'échantillonnage mais plutôt lors de la résolution de l'instance, par exemple à travers une fonction objectif F dans le cas d'une programmation par contraintes – voir section 6.5. Ceci permet de pleinement exploiter ϕ^r.

En résumé, nous avons proposé trois méthodes d'échantillonnage. Dans les deux premières, l'espace des positions accessibles et des positions observables est confondu : seul le premier est échantillonné, en fonction de l'utilité qu'il apporte à travers la fonction de perception sur les positions observables. Ces deux espaces sont donc reliés *a priori*, avant la phase de résolution : les modèles sont sous-exploités ou « mal exploités ». La différence entre les deux formulations tient à cette fonction de perception. La troisième méthode d'échantillonnage distingue, elle, clairement les po-

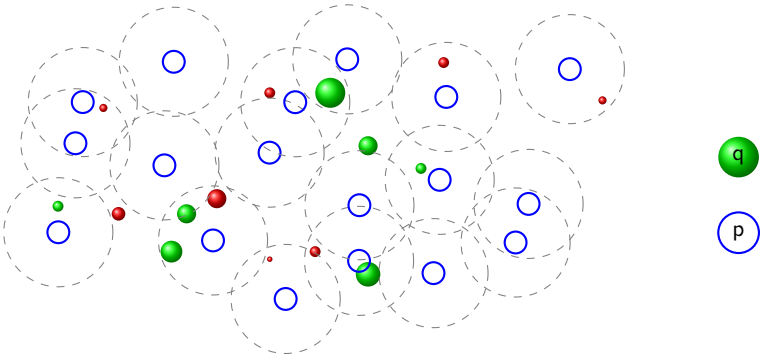


FIGURE 30 – Échantillonnage multiple utilisant un modèle fin de capteur : l'échantillonnage des positions accessibles et celui des positions observables sont effectués indépendamment. Les boules vertes observables sont effectivement échantillonnées, les rouges ne sont pas retenues par l'échantillonnage. Les liens de visibilité ne sont pas encore exploités à ce stade (contrairement aux autres méthodes d'échantillonnage), ce qui permettra de les prendre plus finement en compte, évitant notamment de compter les observations multiples. Les positions accessibles n'ont donc pas encore d'utilité associées : nous choisissons donc de les échantillonner de manière régulière pour couvrir l'espace de recherche des solutions aussi bien que possible.

sitions accessibles et les positions observables ; elle ne fait aucun *a priori* sur les liens que les relient et laisse à la phase de résolution le soin de les déterminer.

Dans les deux sections suivantes, nous nous concentrons sur la résolution des problèmes instanciés. Les modèles sont donc discrets, grâce aux processus d'échantillonnage. De plus, sauf mention contraire comme à la section 6.6.4, nous considérons une parfaite capacité de communication, c'est-à-dire que les déplacements des robots ne sont pas contraints par le besoin de communiquer.

6.4 RÉSOLUTION PAR PARCOURS DE GRAPHE

L'algorithme de Dijkstra est un algorithme classique de calcul de plus courts chemins dans un graphe.

À la fin de l'étape d'échantillonnage, nous obtenons un problème instancié sous forme d'un graphe représentant l'environnement avec les déplacements et les observations possibles – voir la section 6.2. Nous proposons ici de résoudre ce problème instancié à l'aide d'un algorithme de parcours de graphes, inspiré de l'algorithme de Dijkstra*. L'algorithme est donné dans le cas monorobot, mais comme expliqué plus loin il est facile de l'étendre au cas multirobot, en utilisant des variables vectorielles. Cet algorithme nous renvoie une solution optimale, c'est-à-dire ici une solution d'utilité maximale pour un coût d'au plus T. Nous fournissons la preuve de la correction de l'algorithme et commentons sa complexité.

6.4.1 *Algorithme de patrouille*

Nous proposons l'algorithme 1 permettant de définir des schémas de patrouille optimaux sur une instance de problème. C'est un algorithme de parcours de graphe, utilisé sur le graphe $\mathcal{G}$ représentant l'espace des solutions, c'est-à-dire les chemins possibles pour les robots – voir les sections 6.2 et 6.3. Cet algorithme fonctionne pour des chemins de coûts strictement positifs et d'utilité positive. Rappelons que, dans notre cas, nous évaluons le coût d'un chemin en temps : rester stationnaire a donc aussi un certain coût strictement positif.

Le nœud $\mathtt{start\_node}$ désigne le nœud de $\mathcal{G}$ correspondant à la position de départ du robot. On cherche un plan de patrouille $\mathtt{patrol\_plan}$, qui est une chemin constitué d'une suite de points de passage, indiqués par les nœuds du graphe. Ce plan est donné par la fonction MAX_UTIL_DIJKSTRA, qui permet de calculer un ensemble de plans et sous-plans optimaux. Cette fonction utilise en particulier deux ensembles $\mathcal{W}$ et $\mathcal{X}$ contenant des plans, qui sont des chemins dans le graphe $\mathcal{G}(\mathcal{A}, \mathcal{V})$, espace de recherche des solutions. Plus spécifiquement, et à la manière de Dijkstra, $\mathcal{W}$ contient des chemins construits mais pas encore traités par la boucle WHILE, et $\mathcal{X}$ contient les chemins traités par cette boucle – équivalent des nœuds « visités ». Lorsque $\mathcal{W}$ est vide, on a trouvé tous les chemins possibles dans $\mathcal{G}$ de coût maximal, et ils sont rassemblés dans $\mathcal{X}$. On y cherche alors celui d'utilité maximale. La fonction GET_LAST_NODE renvoie le dernier nœud d'un chemin dans $\mathcal{G}$: cela permet d'étendre le chemin. Un chemin est noté $(s_1, s_2, ...s_k)$. Deux tableaux $\mathtt{cost}[]$ et $\mathtt{util}[]$ permettent respectivement de stocker le coût et l'utilité d'un chemin.

Remarque. L'algorithme est donné en monorobot et fonctionne donc avec le graphe $\mathcal{G}^r$. Il est néanmoins facile de l'adapter pour qu'il fonctionne avec le graphe $\mathcal{G}(\mathcal{A}, \mathcal{V})$, produit des graphes $\mathcal{G}^r$. En particulier, il faut utiliser des vecteurs pour représenter les coûts et les positions des robots, chaque arc du graphe représentant l'action d'*un* robot. T, $\mathtt{cost}[u]$, $c_{v,w}$ et $\mathtt{start\_node}$ sont alors des vecteurs de dimension égale au nombre de robots.

6.4.2 *Terminaison, correction et complexité*

Lemme 6.1. *L'algorithme 1 se termine.*

Démonstration. La fonction MAX_UTIL_DIJKSTRA se termine parce que l'invariant $\mathcal{X}$ est un ensemble de taille strictement croissante à chaque étape. Or, $\mathcal{X}$ ne contient que des éléments issus de $\mathcal{W}$, c'est-à-dire des chemins dans $\mathcal{G}$ ayant un coût inférieur à T. Ce dernier nombre étant fini, $\mathcal{X}$ est nécessairement fini, de même que le nombre d'étapes dans l'algorithme. Ce dernier termine donc. □

Algorithme 1 Algorithme de patrouille de type Dijkstra

Input: a graph $\mathcal{G}$, a source $start\_node$
Output: an optimal patrol plan

```
1: function MAX_UTIL_DIJKSTRA(GRAPH, SOURCE)
2:     cost[(source)] ← 0
3:     util[(source)] ← 0
4:     W ← (source)
5:     X empty
6:     while W ≠ empty do
7:         s ← arg min_{s∈W} cost[s]
8:         W ← W \ {s}
9:         X ← X ∪ {s}
10:        v ← GET_LAST_NODE(s)
11:        for each neighbor w of v do
12:            alt ← cost[s] + c_{v,w}
13:            if alt ⩽ T then
14:                t ← (source, ..., v, w)
15:                util[t] ← util[s] + util(v, w)
16:                cost[t] ← alt
17:                W ← W ∪ {t}
18:            end if
19:        end for
20:    end while
21:    plan ← arg max_{s∈X} util[s]
22:    return plan
23: end function
24: return MAX_UTIL_DIJKSTRA(G, start_node)
```

Un algorithme est dit correct lorsque, pour chaque instance du problème, il se termine en produisant la bonne sortie, c'est-à-dire qu'il résout le problème posé.

Théorème 6.2. *L'algorithme 1 est correct*.*

Démonstration. L'algorithme se termine d'après le lemme 6.1. Montrons maintenant que l'algorithme renvoie bien un schéma de patrouille optimal, lorsqu'un tel schéma existe.

Sans perte de généralité (quitte à normaliser les valeurs et à rajouter des nœuds virtuels), nous supposons T entier et les arcs de coûts $c_{v,w}$ unitaires. Montrons qu'à la fin de la boucle WHILE, $\mathcal{X}$ contient tous les chemins de $\mathcal{G}$ commençant en $start\_node$ et de coût au plus T. Tout chemin de longueur $L+1$ – avec $L < T$ entier – commençant en $start\_node$ est nécessairement constitué d'un sous-chemin de longueur L commençant en $start\_node$. Or, un tel sous-chemin s ne peut se trouver en $\mathcal{X}$ que s'il a été traité dans la boucle WHILE, c'est-à-dire après qu'on ait ajouté dans $\mathcal{W}$ tous les chemins constitués de s prolongé d'un arc dans $\mathcal{G}$ par l'intermédiaire de la boucle FOR. Tous ces nouveaux chemins seront alors traités, et d'après les hypothèses, leur longueur $L+1 \leqslant T$ leur garantit d'être ajouté à $\mathcal{X}$. Ainsi, si la propriété $P(L)$ « tout chemin de longueur $L < T$ est dans $\mathcal{X}$ »

est vraie, alors $P(L+1)$ est vraie aussi. $P(0)$ étant trivialement vrai, $P(L)$ est vrai par récurrence pour tout $L \leqslant T$. À la fin de la boucle WHILE, X contient donc bien tous les chemins de $\mathcal{G}$ commençant en $start\_node$ et de coût au plus T.

$\mathcal{X}$ contient donc les solutions optimales au problème instancié : ce sont ses éléments d'utilité maximale. L'une d'elles est renvoyée par l'algorithme, ce qui prouve la correction de ce dernier. □

Propriété 6.1. *L'algorithme 1 a une complexité de calcul maximale de*

$$O\Big(\prod_{r\in R} Tb^r\Big) \tag{9}$$

en notant T *le coût maximal et* b^r *le facteur de branchement du graphe* $\mathcal{G}^r$.

Démonstration. Les fonctions $\arg\min$ et $\arg\max$ sont linéaires en la taille de l'ensemble sur lequel elles sont appliquées. En supposant un coût de déplacement unitaire, et en notant b^r le facteur de branchement maximal du graphe $\mathcal{G}^r$ des positions accessibles du robot r, la valeur $\min_{\mathcal{W}}(cost)$ est un vecteur à N dimensions dont chaque élément représente la distance parcourue par le robot correspondant. Chaque robot peut parcourir au maximum T étapes, et décide pour chaque étape parmi b^r choix, ce qui résulte en Tb^r chemins possibles par robot. On suppose par ailleurs que les fonctions de calcul des coûts et utilité sont en temps constant, ce qui est généralement le cas après d'éventuels précalculs. Il en résulte la complexité de $O(\prod_{r\in R} Tb^r)$ pour la fonction MAX_UTIL_DIJKSTRA, et donc pour l'algorithme. □

Remarque. La complexité en mémoire n'est pas abordée ici, car ce n'est pas le goulot d'étranglement de notre algorithme.

Si l'on considère que $b^r = b$, $\forall r \in R$ (c'est-à-dire que chaque robot a le même facteur de branchement), alors on peut simplifier l'écriture de la complexité de la la manière suivante :

$$O\Big((bT)^N\Big) \tag{10}$$

avec N le nombre de robots. La complexité de calcul est donc exponentielle avec le nombre de robots. Cela rend l'algorithme difficilement utilisable pour des équipes de robots de taille moyenne ou grande, du moins avec un facteur de branchement b et une période T raisonnable. Néanmoins, pour de petites équipes de robots, l'algorithme 1 devrait bien fonctionner.

En outre, par rapport à une solution par recherche de cycle hamiltonien, notre algorithme est capable de gérer des niveaux de priorités arbitraires entre les nœuds, cela à travers les utilités. Avec une faible adaptation, il peut également gérer des utilités dynamiques. En effet, il suffit d'utiliser une fonction $util[s]$ dépendant du temps et des positions déjà observées, à l'aide d'un marquage de ces positions. L'algorithme fonctionne en outre pour tout type de graphe, sous réserve que les coûts des arcs soient strictement positifs

et les utilités positives, ce qui est presque toujours le cas en pratique. Le résultat de complexité incite néanmoins à trouver d'autres approches, ce que nous faisons dans la section suivante.

6.5 OPTIMISATION EN NOMBRES ENTIERS

Dans cette section nous formulons le problème de patrouille instancié comme un problème de satisfaction de contraintes, sur lequel on va chercher à optimiser une valeur objectif. Plus précisément, on va résoudre le problème par optimisation en nombres entiers – ou IP – et, si possible, avec une formulation linéaire – *Mixed Integer Linear Programming* (MILP). Ces dernières sont en effet généralement plus faciles à résoudre. Dans le cas général, il s'agit donc de résoudre le problème suivant sur l'ensemble des plans possibles X :

$$\min F(X) \text{ tel que } G(X) \leqslant 0 \text{ et } H(X) = 0 \tag{11}$$

Dans ce type de formulation, F est une fonction à optimiser – ici minimiser – comme une somme de coûts par exemple. G et H sont des contraintes à respecter – comme une vitesse maximale – restreignant les solutions X valides. Les différents modèles de perception impactent directement l'espace de recherche $\{X\}$ et la fonction objectif F tandis que les communications impactent les contraintes G et H – voir la section 6.6.4. Nous considérons ici trois formulations IP différentes, chacune correspondant à une des méthodes d'échantillonnage présentées en section 6.3 : (i) un modèle basique, donnant une formulation proche du TSP, (ii) un modèle plus élaboré mais exploité grossièrement, donnant un formulation très proche de (i), et enfin (iii) un modèle élaboré pleinement exploité, sans approximation, donnant un problème d'optimisation plus complexe, avec notamment de nouvelles variables.

Un problème d'optimisation formulé comme (11) est dit non linéaire si au moins F, G ou H n'est pas linéaire.

La linéarisation a généralement un coût – des variables additionnelles par exemple – mais ce dernier reste en principe inférieur au bénéfice qu'on en tire.

Nous décrivons un plan X comme un ensemble $X = \{x_{ij}^r \in \{0,1\}, r \in R, (p_i^r, p_j^r) \in {P_r}^2, a_{ij}^r = 1\}$. Autrement dit, un plan est constitué d'un ensemble de chemins valides, un par robot. $x_{ij}^r = 1$ indique que le robot r se déplace de p_i^r à p_j^r lors de son chemin de patrouille. Le problème que nous posons est non linéaire* car on s'attend à la présence de fonctions min et/ou max dans les objectifs F ou les contraintes G et H. Nous pouvons choisir de résoudre le problème à l'aide de solveurs non-linéaires. Néanmoins, X étant fait de variables binaires, nous essayons autant que possible de reformuler le problème de manière linéaire comme un *Binary Linear Problems*, c'est-à-dire comme un cas particulier des problèmes MILP. En effet, les solveurs linéaires sont en règle générale plus efficaces que les solveurs non linéaires et permettent d'aborder de plus larges instances de problèmes. La linéarisation implique que F, G et H soient toutes trois linéaires et nécessite pour cela de transformer certaines contraintes, généralement à l'aide de variables additionnelles*.

En outre, en supposant – sans perte de généralité – que $t = 0$ au début de la phase de planification, nous considérons que l'horizon de planification T est tel que l'équation suivante est vraie tant qu'aucune observation n'est faite :

$$u(q,t) - u(q,0) \ll u(q,0) \qquad \forall q,\ \forall t \in [0,T] \tag{12}$$

En d'autres termes, l'utilité d'une position observable croît faiblement durant l'horizon temporel considéré et par rapport à la valeur d'utilité au début de la phase de planification. Cela signifie que nous pouvons considérer l'utilité des positions observables comme constantes pour la phase de planification en cours. Ce raisonnement permet de réduire drastiquement la complexité du problème instancié, car les fonctions F, G et H ne sont plus dépendantes du temps t. Pour la suite, nous utilisons donc :

$$u(q,t) \approx u(q) = u(q,0) \qquad \forall t \in [0,T] \tag{13}$$

Bien entendu, une tâche d'observation réduit cette utilité. Comme nous nous concentrons sur un problème instancié et un horizon temporel de T seulement, nous ne considérons plus les variations temporelles, sauf mention contraire. Q étant discret, nous utilisons la notation $u_k = u(q_k)$. L'hypothèse d'indépendance par rapport au temps ne permet pas de linéariser le problème, mais la suppression cette dépendance réduit drastiquement le nombre de variables, ce qui simplifie sensiblement la résolution du problème – voir aussi la section 6.6.2 à ce sujet.

6.5.1 *Formulation TOP orientée positions*

La formulation proposée ci-dessous est inspirée de la formulation par flot du problème de TSP. Plus spécifiquement, c'est une formulation par flot du *Team Orienteering Problem* (TOP) [38, 170]. Un *Orienteering Problem* (OP)* consiste à déterminer le chemin hamiltonien d'un sous-ensemble de V, incluant des sommets de départs et éventuellement d'arrivée fixés, et n'ayant pas une longueur supérieure à T. Le sous-ensemble est choisi de manière à maximiser une valeur collectée – ici l'utilité – et déterminer ce meilleur sous-ensemble fait partie du problème, ce qui rend un OP plus complexe qu'un simple TSP – il y a deux problèmes difficiles en un. Un TOP est la version multi-agents d'un OP. Nous considérons ici une variante dans laquelle les sommets d'arrivée ne sont pas fixés au préalable. Nous la notons TOP-pos, car elle s'appuie sur le processus d'échantillonnage présentée à la section 6.3.1 et orienté vers les positions des robots.

"The OP can be seen as a combination between the Knapsack Problem (KP) and the Travelling Salesperson Problem (TSP)." – Vansteenwegen [170]

Considérant cet étape d'échantillonnage, nous désirons résoudre le problème suivant :

$$\max \quad \sum_{j \in V} u_j y_j \tag{14}$$

Sous contraintes

$$y_j = \min(1, \sum_{r \in R, (i,j) \in \mathcal{A}^r} x_{ij}^r) \qquad \forall j \in \mathcal{V} \qquad (15)$$

$$\sum_{j:(i,j) \in \mathcal{A}^r} x_{ij}^r - \sum_{j:(j,i) \in \mathcal{A}^r} x_{ji}^r = d_i^r \qquad \forall r \in R, \forall i \in \mathcal{P}^r \qquad (16)$$

$$\sum_{i \in \mathcal{P}^r} d_i^r = 0 \qquad \forall r \in R \qquad (17)$$

$$\sum_{(i,j) \in \mathcal{A}^r} c_{ij}^r x_{ij}^r \leqslant T \qquad \forall r \in R \qquad (18)$$

Rappelons tout d'abord que les variables x_{ij}^r sont binaires ; les nouvelles variables y_j^r le sont aussi. (14) indique que nous voulons rassembler autant de données – utilité – que possible, l'utilité étant gagnée lorsque les positions associées sont observées.

(15) exprime le fait que la position p_j est visitée par l'équipe si au moins un robot l'atteint, et qu'il n'est pas utile de la visiter plus d'une fois, ceci en accord avec les modèles de capteur – les observations sont ici considérées comme parfaites – et avec le fait que u est considérée comme constante sur la période T.

(16) utilise des variables additionnelles $D^r = \{d_i^r : p_i^r \in \mathcal{V}^r\}$ où d_i^r est la « demande nette » au nœud p_i^r ; $d_i^r = -1$ pour le nœud de départ (la position de départ est le nœud numéro 1), $d_i^r = +1$ pour le dernier nœud, et $d_i^r = 0$ pour les nœuds intermédiaires, de transit. Les nœuds finaux n'étant ici spécifiés pour aucun des robots, les d_i^r sont des variables binaires additionnelles et non des paramètres, à l'exception des nœuds de départ. Leur somme est égale à 1 pour chaque robot (17).

(18) prend en compte le coût des chemins pour les robots, et la limite de temps qui les contraint.

(15) est non linéaire, ce qui empêche d'utiliser une formulation MILP comme désiré. Néanmoins, en remarquant que nous voulons maximiser les $(y_j)_{j \in V}$, nous pouvons remplacer (15) par les inégalités linéaires suivantes afin d'obtenir, comme désiré, une formulation MILP :

$$y_j \leqslant 1 \qquad \forall j \in \mathcal{V} \qquad (19)$$

$$y_j \leqslant \sum_{r \in R, (i,j) \in \mathcal{A}^r} x_{ij}^r \qquad \forall j \in \mathcal{V} \qquad (20)$$

Notons que cela n'impacte pas significativement le nombre de contraintes, et (19) peut être avantageusement remplacée en spécifiant directement que les $(y_j)_{j \in \mathcal{V}}$ doivent être des variables binaires. De plus, avec les $(y_j)_{j \in \mathcal{V}}$ en variables binaires, (20) peut être redéfinie comme une égalité, imposant que le nœud p_j ne soit visité qu'une seule fois par l'équipe. De cette manière, on s'interdit de visiter plusieurs fois un nœud, ce qui contraint un peu plus l'espace de recherche, mais permet de faciliter la recherche de solutions.

6.5.2 *Formulation TOP orientée perceptions*

Considérant l'étape d'échantillonnage décrite en 6.3.2, nous désirons résoudre un problème presque identique à celui décrit à la section précédente :

$$\max \quad \sum_{j \in \mathcal{V}} v_j y_j \tag{21}$$

vérifiant les mêmes contraintes que la fonction objectif (14). En effet, les deux formulations ne diffèrent que par les utilités – v_j au lieu de u_j – et par les points échantillonnés. Cela peut influencer la qualité des solutions résultantes mais pas les autres considérations : méthodes de résolution, complexité, linéarisation des contraintes, contraintes additionnelles, *etc.*.

La formulation présentée ici et notée TOP-per est presque identique à la formulation TOP-pos présentée section 6.5.1, mais les solutions peuvent être aussi différentes que le sont les données d'entrées issus de l'étape d'instanciation. En fait, on peut considérer la formulation TOP-pos comme un cas particulier et dégénéré de la formulation TOP-per, dans lequel les fonctions de perception ϕ^r sont des Dirac centrés sur les positions des robots. À cause des défauts dans l'exploitation des modèles de capteurs mentionnés dans la section 6.3.2, il n'y a pas de garantie que les solutions résultantes soient meilleures qu'avec la résolution de (14). On peut cependant estimer cela probable, dans la mesure où certaines positions p_j permettent de percevoir plusieurs positions observables q, ce qui est ici pris en compte mais ignoré dans la section précédente.

6.5.3 *Formulation « Sightseeing Problem »*

Dans la troisième méthode d'échantillonnage présentée à la section 6.3.3, les déplacements et la perception sont mieux distingués, et l'espace des solutions se base sur l'échantillonnage des deux types de positions : celles accessibles par les robots et celles observables. Cela change la formulation du problème de la manière suivante :

$$\max \quad \sum_{q_k \in Q} u_k y_k \tag{22}$$

Sous les contraintes :

$$y_k = \max_{r \in R, (i,j) \in \mathcal{A}^r} (0, \phi^r_{jk} x^r_{ij}) \qquad \forall k \in Q \tag{23}$$

$$\sum_{j:(i,j) \in \mathcal{A}^r} x^r_{ij} - \sum_{j:(j,i) \in \mathcal{A}^r} x^r_{ji} = d^r_i \qquad \forall r \in R, \forall i \in \mathcal{P}^r \tag{24}$$

$$\sum_{i \in \mathcal{P}^r} d^r_i = 0 \qquad \forall r \in R \tag{25}$$

$$\sum_{(i,j) \in \mathcal{A}^r} c^r_{ij} x^r_{ij} \leqslant T \qquad \forall r \in R \tag{26}$$

Cette formulation n'est plus une simple variation du TSP ou du TOP. En effet, comparons cette nouvelle formulation à la formulation TOP : tout comme (14), (22) exprime que nous voulons rassembler autant de données que possible. Cependant, et contrairement à (15), (23) indique que pour chaque position observée $q_k \in Q$ nous prenons seulement en compte la meilleure observation parmi toutes celles possibles, c'est-à-dire que seule est prise en compte la position visitée p_j^r depuis laquelle l'équipe a la meilleure valeur de perception ϕ_{jk}^r sur la position observée. Une position p_j^r est dite visitée par le robot r ssi $x_{ij}^r = 1$.

Ceci diffère des autres formulations par le fait qu'ici les observations ne sont plus du tout indépendantes les unes des autres et se retrouvent au contraire comme « en compétition » les unes avec les autres.

Nous désignons cette formulation sous le terme de « Sightseeing Problem » (SP) en référence aux activités touristiques. En effet, le but est d'observer les lieux – positions – les plus attractifs, en trouvant le meilleur compromis entre la qualité du point de vue et sa difficulté d'accès, qui limite le nombre de points touristiques que l'on pourra visiter.

(24), (25) et (26) sont identiques à (16), (17) et (18) respectivement. Tout comme (15), (23) n'est pas linéaire. Sa linéarisation est un peu plus compliquée et astucieuse que pour (15), et nécessite l'introduction d'une constante M – arbitraire mais de valeur suffisamment élevée – et des variables binaires b_{jk}^r $\forall q_k \in Q$, $\forall r \in R$, $\forall j \in P_r$. En se rappelant que l'on cherche à maximiser les $(y_k)_{q_k \in Q}$, il est possible de remplacer (23) par les contraintes suivantes $\forall r \in R, p_j \in P^r, \forall q_k \in Q$:

$$0 \leqslant y_k \leqslant \Big(\sum_{i:(i,j)\in A^r} x_{ij}^r \Big) \phi_{jk}^r + (1 - b_{jk}^r) M \tag{27}$$

tandis que $(b_{jk}^r)_{r\in R, j)\in P^r}$ sont tels que :

$$\sum_{r\in R, j\in P^r} b_{jk}^r = 1 \qquad\qquad \forall q_k \in Q \tag{28}$$

(28) signifie que seul un des b_{jk}^r est égal à 1, les autres étant égaux à zéro. Ainsi, (27) n'est une contrainte réelle que pour le seul b_{jk}^r non nul. Comme nous souhaitons maximiser les y_k, le solveur choisira les b_{jk}^r de manière à satisfaire au mieux l'objectif, se comportant ainsi comme un équivalent de la fonction $\max$. Les b_{jk}^r seront en effet sélectionnés pour désigner la meilleure observation de q_k, faite par le robot r à la position $p_j \in P^r$.

Remarque. Cette formulation linéarisée nécessite de nombreuses variables additionnelles, ce qui peut devenir une vraie difficulté pour les solveurs. On peut alors se poser la question la pertinence de la linéarisation, face à l'utilisation d'un solveur non linéaire, comme discuté à la section 6.6.1 et en annexe 6.6.2.

6.5.4 *Élimination des sous-tours*

Un problème bien connu dans le problème du TSP est l'apparition de « sous-tours » tels qu'illustrés à la figure 31. Ces sous-tours sont problématiques car ils mènent à des solutions inapplicables en pratique. Ce problème survient également dans les variantes du TSP, ce qui inclut les *Team Orienteering Problems*. Les formulations proposées précédemment – sections 6.5.1, 6.5.2 et 6.5.3 – n'empêchent pas la présence de sous-tours dans les solutions. Il est nécessaire de leur ajouter des contraintes supplémentaires.

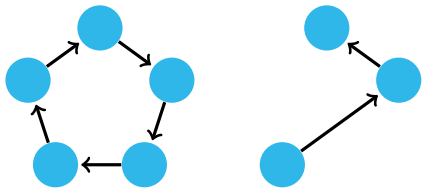


FIGURE 31 – Exemple de problème de sous-tours pour une solution d'un TSP monorobot : sans contraintes additionnelles, les solutions peuvent contenir des sous-tours, c'est-à-dire que les chemins peuvent être constitués de parties disjointes.

6.5.4.1 *Contraintes MTZ*

Il existe plusieurs jeux de contraintes permettant de prévenir l'apparition de sous-tours dans les solutions, appelées *Subtours Elimination Constraintes* (SEC). Nous avons choisi d'utiliser les SEC de Miller-Tucker-Zemlin (MTZ) [104]. Ce sont des contraintes classiques, reconnues pour leur clarté et leur efficacité.

$$\beta_i^r - \beta_j^r + 1 \leqslant (n^r - 1)(1 - x_{ij}^r) \qquad \forall r \in R,\ \forall (i,j) \in A^r \tag{29}$$

(29) introduit un nouvel ensemble de $\{\beta_i^r : i \in \mathcal{V}^r, i \neq 1\}$. Les β_i^r sont des nombres réels arbitraires mais ils peuvent êtres reclassés en nombres entiers positifs représentant l'ordre de visite des nœuds. Par commodité, nous ajoutons la contrainte $\beta_1^r = 1\ \forall r \in R$ – le nœud 1 étant considéré comme la position de départ – et nous limitons les valeurs possibles pour les β_i^r, cela dans le but d'aider les logiciels d'optimisation [131] :

$$2 \leqslant \beta_i^r \leqslant n^r \qquad \forall r \in R,\ \forall i \in \mathcal{V}^r,\ i \geqslant 1 \tag{30}$$

Nous ajoutons donc ces contraintes MTZ à nos formulations précédentes*– sections 6.5.1, 6.5.2 et 6.5.3. Elles ont les propriétés suivantes [145] :

Un même jeu de SEC est valable pour différentes types de problèmes (TSP, TOP, etc.*).*

- le nœud 1 est la position de départ ;
- chaque position p_i^r visitée est connectée à un tour lui-même relié aux positions de départ, ce qui permet d'éliminer les sous-tours ;
- il n'est pas nécessaire que tous les nœuds i soient visités, à moins que d'autres contraintes ne l'imposent ;

— les $(\beta_i^r)_{i \in \mathcal{V}^r}$ peuvent être vues comme une borne supérieure de l'ordre de visite du nœud i, lorsqu'il est visité.

Le principal inconvénient des contraintes MTZ dans le cas général du TSP sont qu'elles nécessitent de choisir des positions de départ. Dans notre cas, cela n'est pas un problème puisque ces dernières sont connues et même imposées. Il peut également y avoir des problèmes de faisabilité dans des graphes non complets, lorsque toute solution nécessite de visiter au moins un nœud plusieurs fois. La résolution avec contraintes MTZ n'échoue (en théorie) jamais lorsque les solutions optimales ne visitent chaque nœud qu'une fois, et *a fortiori* si le graphe est complet. Il est important d'être conscient de ces limitations, bien qu'en pratique cela s'avère rarement problématique – voir le chapitre 7.

6.5.4.2 *Contraintes alternatives*

Nous avons fait le choix des contraintes MTZ qui nous ont donné satisfaction en pratique – voir les chapitres 7 et 8. Bien que très répandues, il existe néanmoins des réserves à leur sujet. Ainsi, Letchford et collab. soulignent que "*the MTZ formulation is compact, having only* $O(n^2)$ *variables and* $O(n^2)$ *constraints. Unfortunately, Padberg & Sung show that its LP relaxation yields an extremely weak lower bound, much weaker than that of the DFJ formulation*" [95]. En d'autres termes, les SEC MTZ ne sont pas les plus efficaces du fait que la plupart des solveurs MILP utilisent la relaxation* comme une heuristique ou une borne de la solution finale.

La relaxation d'un problème IP consiste à le résoudre dans l'espace des réels plutôt que dans celui des entiers.

Outre certains jeux de SEC connus et nommés tels que DFJ ou MCF, une variante appréciée consiste à introduire des SEC *ad hoc* « en court de route », c'est-à-dire au fur et à mesure de processus de résolution. Ce processus est défini par l'algorithme 2. À première vue, on pourrait croire ce processus moins efficace car il nécessite de résoudre le problème de multiples fois. En pratique, il s'avère pertinent car les problèmes sont résolus plus rapidement car étant moins complexes ; en particulier, le nombre de variables et de contraintes est drastiquement restreint.

Algorithme 2 Algorithme d'ajout de SEC dynamique

Input: a TOP instance
Output: an optimal TOP solution (with connex paths)
1: define current model as (14), (15), (16) and (18)
2: **while** optimal solution not found **do**
3: solve the current model to optimality by an ILP-solver
4: **if** solution contains no subtour **then**
5: set the solution as the optimal one
6: **else**
7: find all subtours of the solution
8: add the corresponding subtour constraints into the model
9: **end if**
10: **end while**

6.6 DISCUSSION SUR LES FORMULATIONS IP

Après avoir présenté trois formulations IP différentes à la section 6.5, nous discutons ici de la façon de les résoudre, de fonctions objectifs alternatives, de l'ajout de contraintes de communication, et de l'aspect cyclique des problèmes de patrouille.

Remarque. Nous avons précédemment indiqué que la complexité de notre algorithme 1 – section 6.4 – gênait son application pratique. Avec la méthode de résolution par optimisation en nombres entiers proposée à la section 6.5, il n'est pas non plus envisageable de résoudre optimalement de larges instances de problèmes. Cependant, ce type de formulation permet également des résolutions approchées en utilisant des techniques d'optimisation stochastique, ce que l'algorithme de parcours de graphe proposé ne permet pas. Les formulations IP présentent donc une souplesse supplémentaire, et pour cette raison la suite de nos travaux s'est pleinement concentrée sur celles-ci.

6.6.1 *Exploitation des modèles*

Nous avons différentes formulations IP permettant d'élaborer des schémas de patrouille, sur la base du formalisme décrit à la section 6.2. Il nous semble important de bien comprendre à ce stade les différences entre les formulations proposées et à quels niveaux interviennent les différents modèles (déplacements, capteurs, utilité)*. Cela est résumé par le tableau 2.

Les détails des fonctions utilisées dans la partie expérimentale sont présentés à la section 7.1.

	Échantillonnage		Résolution	
	Q	P^r	Contraintes	Objectif
TOP-pos	N/A	u	c^r	u
TOP-per	N/A	v	c^r	v
SP	u	-	c^r	$\max \phi^r.u$

TABLE 2 – Récapitulatif des modèles utilisés par les différentes formulations IP présentées tout au long de ce chapitre ; pour rappel, on a la relation $v = \sum \phi^r.u$.

On constate que les trois formulations utilisent les modèles de déplacement en intégrant les coûts c^r de ces derniers comme des contraintes, et que l'utilité u est intégrée à la fonction objectif, parfois conjuguée à la perception ϕ^r.

Les formulations TOP-pos et TOP-per sont très similaires : les positions observables Q ne sont pas échantillonnées car c'est directement l'échantillonnage des positions accessibles P^r qui exploite les utilités à travers un modèle de visibilité. C'est également la principale différence avec la formulation SP présentée en section 6.5.3. Cette dernière exploite en effet les ϕ^r

au niveau de la résolution du problème, et non lors de l'échantillonnage, ce qui nous oblige à bien distinguer et échantillonner séparément Q et les P^{r}.

Ce double échantillonnage et les variables et contraintes additionnelles induites font augmenter drastiquement la complexité du problème IP. En conséquence , la taille des instances pouvant être résolues est réduite par rapport aux autres formulations plus simplistes – voir aussi la section 6.6.2 et le chapitre 7 présentant les différents résultats.

6.6.2 *Résoudre le problème d'optimisation*

Il existe de nombreuses façons de résoudre les problèmes d'optimisation en nombres entiers comme ceux que nous avons formulés. En particulier, on compte de nombreux solveurs libres ou commerciaux permettant de résoudre des problèmes IP linéaires (ILP ou MILP), quadratiques (IQP) ou non linéaires (INLP). Ces solveurs tentent de résoudre les problèmes IP de manière optimale, et proposent des APIs en différents langages (C, C++, Python, *etc.*). Il est également possible d'utiliser des méthodes de résolutions stochastiques telles que les algorithmes génétiques ou l'entropie croisée qui donnent très souvent des solutions viables, mais ne peuvent garantir un optimum global. Enfin, il est possible d'élaborer un solveur spécifique au problème formulé : l'étude de Vansteenwegen propose notamment à cet effet une liste d'heuristiques pour algorithmes stochastiques permettant de résoudre les instances d'*Orienteering Problems* [170].

Nous avons choisi de privilégier les formulations linéaires de nos problèmes, et de nous concentrer sur une seule méthode – classique – de résolution des formulations MILP, à l'aide du solveur GLPK. Ce choix est justifié en annexe B ; celle-ci contient également plusieurs éléments de comparaison entre les différentes méthodes de résolution, et le lecteur intéressé est renvoyé à sa lecture bien qu'elle ne soit pas indispensable pour comprendre la suite de notre discours. Soulignons néanmoins ici que ce qui nous importe est de montrer que ces formulations IP sont viables dans l'élaboration de schémas de patrouille en contexte antagoniste. Notre objectif n'est pas d'élaborer de nouvelles méthodes de résolution de ces problèmes, ni de les comparer entre elles.

En outre, nous étudions deux grandes approches : une approche centralisée, résolvant optimalement le problème, et une approche séquentielle que nous voulons utiliser de manière décentralisée. Cette dernière approche est sous-optimale mais permet de réduire la complexité du problème et donc de traiter des instances plus larges : l'idée principale est que les robots planifient l'un après l'autre, pour eux-mêmes seulement, chaque robot faisant ainsi face à un problème de taille réduite, par exemple un « simple » OP au lieu d'un TOP. Les résultats respectifs des approches centralisée et séquentielle-décentralisée sont donnés aux chapitres 7 et 8 respectivement.

6.6.3 *Fonctions objectifs alternatives*

Les fonctions objectifs considérées dans les sections 6.5.1 à 6.5.3 sont basées sur la maximisation d'une somme de données collectées, dont la valeur est évaluée par l'utilité gagnée. En d'autres termes, on cherche à réduire autant que possible l'oisiveté *moyenne* sur l'ensemble de la zone. Il est cependant possible de définir d'autres objectifs. Une alternative intéressante est de chercher à minimiser l'oisiveté maximale sur la zone. En d'autres termes, on veut limiter les « pics » d'utilité, ce qui donne une fonction objectif de la forme :

$$\min \quad \max_{j \in \mathcal{V}} \quad u_j \bar{y}_j \tag{31}$$

avec $\bar{y}_j = (1 - y_j)$ indiquant si p_j n'a *pas* été visité (le contraire de y_j donc). Dans le cas de la formulation SP, il faut remplacer $j \in \mathcal{V}$ par $k \in Q$: les y_k correspondent alors à la meilleure observation de q_k, et $\bar{y}_k$ peut être interprété comme l'information restant à découvrir – la différence entre l'observation réelle $y_k \in [0, 1]$ et une observation parfaite égale à 1.

Fondamentalement, ce changement d'objectif ne remet pas en question les propos précédents : en particulier, il incite toujours à faire tendre les y_j vers 1 en les maximisant, ce qui rend valides nos propos sur la linéarisation de certaines contraintes.

Les objectifs de type « $\min\max$ » (« $\max\min$ ») sont relativement bien connus. Ils sont couramment appelés *minimax* (*maximin*) et on les réécrit habituellement sous la forme suivante, introduisant une nouvelle variable Y :

$$\min \quad Y \tag{32}$$

Accompagnée des contraintes :

$$Y \geqslant u_j(1 - y_j) \qquad \forall j \in \mathcal{V} \tag{33}$$

La variable additionnelle Y émule la fonction maximum : en effet, on cherche à la minimiser, et elle ne peut être inférieure aux $\left(u_j(1 - y_j)\right)_{j \in \mathcal{V}}$: elle sera donc égale au plus grand $u_j(1 - y_j)$.

Le problème résultant est plus complexe à cause de l'ajout de nombreuses contraintes – une par variable d'origine – mais ce changement impacte surtout le profil des solutions : il n'est en effet plus possible de « négliger » des positions isolées pour se concentrer sur des groupes de positions spatialement proches et utiles collectivement.

Il est également possible d'introduire dans la fonction objectif un terme incitant à minimiser le coût des chemins :

$$\max \quad \sum_{j \in V} u_j y_j - \quad K * \sum_{(i,j) \in A^r \forall r \in R} c_{ij}^r x_{ij}^r \tag{34}$$

avec K une constante positive permettant d'ajuster les deux objectifs entre eux. Dans notre cas, cette nouvelle fonction objectif présente cependant un intérêt limité car nous nous concentrons sur l'impact de l'équipe de robot lors d'une phase de patrouille de durée fixe, et non sur des contraintes d'économie d'énergie ou sur un compromis temps de patrouille / impact de la patrouille.

6.6.4 *Contraintes de communication*

Nous avons indiqué précédemment que la section 6.5 présupposait l'existence de liens de communication entre les robots, négligeant de fait tous les problèmes relatifs à cet aspect. Nous essayons ici de réellement prendre en compte les communications à l'aide des modèles décrits au début du chapitre. Nous discutons l'impact de ces modèles sur les formulations et notamment leurs complexités.

Nous cherchons spécifiquement des communications périodiques, de période T correspondant à chaque cycle de planification et à la durée des plans des robots. Cela signifie que nous imposons seulement que les robots soient en communication entre eux au début et à la fin de chaque cycle de patrouille, agissant de manière autonome, sans communication, durant les cycles. Ces phases de communication ont pour objectif de partager des données pour se coordonner au mieux pour la mission de patrouille ou de rapporter les résultats de la patrouille à un opérateur. Nous voulons en outre que toute l'équipe soit connectée ensemble, c'est-à-dire que la communication soit possible entre toute paire de robots, éventuellement à l'aide d'autres robots membres de l'équipe comme relais de communications. En termes mathématiques, cela signifie que le graphe de communication* de l'équipe doit être connexe mais il n'est pas nécessaire qu'il soit complet.

Un graphe de communication rend compte des liens de communication directs *et deux à deux existant au sein d'un ensemble de positions ou d'agents.*

Nous fournissons et détaillons ci-après deux types de contraintes pour l'équipe de patrouille : les premières supposent que les positions finales des robots leur permettant de communiquer sont déterminées par un processus extérieur, voire spécifiés par la mission elle-même – section 6.6.4.1. Ce type de contraintes peut par exemple imposer de retourner aux positions de départ, ou de se regrouper autour de stations de base permettant de recharger des batteries et de communiquer. Les secondes considèrent que les positions finales sont seulement contraintes par les liens de communication mais le choix de ces positions est laissé au programme d'optimisation. C'est alors la résolution de l'IP qui permet de définir les positions adéquates pour chaque robot – section 6.6.4.2. Cette seconde approche est plus simple et plus attrayante mais aussi plus complexe à mettre en œuvre que la première si l'on y néglige le processus externe de détermination des positions finales.

6.6.4.1 *Positions finales contraintes*

Lorsque les positions des robots en fin de cycle sont imposées, il suffit de modifier les $(d_i^r)_{r\in R}$ afin de refléter ces contraintes de manière appropriée. On note p^r_{final} la position finale désirée pour le robot r. Les $(d_i^r)_{r\in R}$ ne

sont alors plus des variables binaires mais des paramètres dont la valeur varie suivant la position du nœud correspondant : $+1$ lorsque $i = final$, -1 lorsque $i = 1$ (les positions de départ), et 0 dans les autres cas. La contrainte (16) (p. 106) tient toujours.

Notons qu'il est ici supposé que les positions finales données respectent les contraintes de communication. En outre, même sous l'hypothèse des communications parfaites et sans contraintes, il peut être intéressant d'imposer des positions finales : ce sont alors d'autres considérations qui fixent celles-ci, par exemple une base de rechargement ou un point de ralliement.

Contrairement à ce qu'on pourrait croire au premier abord, imposer des positions finales permet de réduire l'espace de recherche des solutions en diminuant le nombre de variables à déterminer. Il est donc attendu un résultat bénéfique sur les performances du solveur, sous réserve que ces positions permettent l'existence de solutions valides. Pour autant, choisir de bonnes positions finales est difficile en soi : si la solution retenue sera sans doute optimale par rapport à ces nouvelles contraintes, rien ne garantit que le résultat final sur la mission de patrouille – et non sur la seule instance – soit pertinent.

6.6.4.2 *Modèle de communication amélioré*

Dans cette section, nous proposons de considérer les communications directement dans le problème d'optimisation, comme des contraintes supplémentaires nécessitant également des variables additionnelles. On utilise pour cela le modèle de communication introduit à la section 6.2. Considérant la signification des $(d_i^r)_{r\in R}$ qui permettent entre autres de désigner les positions finales des robots, nous désirons ajouter les contraintes suivantes :

$$d_i^{r_1} = 1 \implies \exists r_2 \in R\backslash\{r_1\} : d_j^{r_2} = 1 \text{ et } \alpha_{ij}^{r_1}\alpha_{ji}^{r_2} = 1 \quad \forall r_1 \in R \tag{35}$$

En d'autres termes, un robot doit choisir une position de fin de cycle lui permettant de communiquer avec au moins un autre robot arrivé à sa dernière position lui aussi.

L'utilisation de liens logiques tels que $\implies$ rentre dans le cadre de la programmation par contraintes mais pas dans celui plus restreint de la programmation en nombres entiers (IP).

Comme l'opérateur logique $\implies$ ne peut être utilisé comme tel dans un problème IP*, nous essayons de changer les prédicats ci-dessus en contraintes adaptées. Nous nous appuyons en particulier sur l'équivalence de la relation logique $A = (B \implies C)$ avec la relation $A = (\neg B \vee C)$.

$$(1 - d_i^{r_1}) + \sum_{\substack{r_2\in\backslash\{r_1\}\\ j\in\mathcal{V}^{r_2}\backslash\{1\}}} d_i^{r_1} d_j^{r_2} \alpha_{ij}^{r_1}\alpha_{ji}^{r_2} \geqslant 1 \quad \forall r_1 \in R\ \forall i \in \mathcal{V}^{r_1} \tag{36}$$

Expliquons cette équation : pour chaque robot r_1 considéré, l'équation (36) doit être vérifiée pour chaque position accessible i. Le second terme de la somme étant positif ou nul, l'équation est trivialement vraie lorsque $d_i^{r_1} = 0$ ou -1, c'est-à-dire lorsque la position étudiée n'est pas la position

finale i_f de r_1. Il nous reste donc à étudier ce dernier cas, pour lequel $d_i^{r_1} = 1$. (36) est alors équivalent à

$$\sum_{\substack{r_2 \in \setminus \{r_1\} \\ j \in \mathcal{V}^{r_2} \setminus \{1\}}} d_j^{r_2} \alpha_{i_f j}^{r_1} \alpha_{j i_f}^{r_2} \geqslant 1 \tag{37}$$

Or, $d_j^{r_2} = 0$ pour $j \in \mathcal{V}^{r_2} \setminus \{1\}$ sauf si $j = j_f$, la position finale de r_2. En conséquence, l'équation se vérifie s'il existe au moins un robot r_2 tel que, depuis sa position finale j_f, on a $\alpha_{i_f, j_f}^{r_1} \alpha_{j_f, i_f}^{r_2} = 1$, c'est-à-dire s'il existe bien un lien de communication entre r_1 et r_2 à leur position finale respective.

Malheureusement, ce type de formulation ne garantit en réalité qu'une seule chose : qu'aucun robot ne sera isolé à la fin du cycle. Cela veut dire que, de manière très similaire aux sous-tours pouvant apparaître dans les solutions de TSP, nous pouvons ici obtenir des sous-groupes de robots sans lien de communication intergroupes. Ces nouvelles contraintes ne répondent donc que partiellement à nos exigences, et il est nécessaire de leur adjoindre des contraintes de types SEC reliant les positions finales elles – cf. section 6.5.4.

Remarque. Comme à la section 6.5.4, il existe plusieurs jeux de SEC alternatives : nous pouvons utiliser des SEC dérivées des SEC MTZ en s'appuyant sur l'équivalence « i est la position finale du robot r ssi $d_i^r = 1$ », ou bien utiliser des SEC dynamiques, construites par essai / erreur comme expliqué par l'algorithme 2.

6.6.4.3 *Communications en planification séquentielle*

Lorsque l'on aborde le problème de manière séquentielle avec les robots planifiant l'un après l'autre, il est plus facile d'introduire des contraintes de communications.

Dans un premier temps, nous proposons de laisser le premier robot planifier sans contrainte son chemin de patrouille. À la suite de quoi les autres robots tentent, l'un après l'autre, de se rattacher aux robots ayant déjà planifié, et dont les positions finales sont donc connues. Ce processus est identique dans l'idée à celui proposé par Hollinger [72] pour les problèmes de fouille – voir pour rappel la figure 8 p. 29. Ce processus fonctionne bien dans l'ensemble mais il n'est pas garanti qu'une solution existe : en effet, le premier robot n'étant pas contraint, il peut s'éloigner des autres de telle manière que les communications ne puissent être rétablies. Ce problème se pose d'ailleurs avec chacun des robots, aucun d'eux n'étant contraint par les robots suivants n'ayant pas encore planifié. En pratique, ce phénomène n'est malheureusement pas rare (voir la section 8.4), d'autant plus que la fonction objectif pousse bien souvent les robots à se disperser pour collecter plus d'utilité.

Afin de résoudre ce problème, nous proposons d'évaluer au préalable la liste – ou du moins un sous-ensemble – des N-uplets de positions finales valides, c'est-à-dire respectant les contraintes de communication. Les ro-

bots sont alors contraints dans leur planification par ces N-uplets, ce qui garantit de garder au moins une solution valide – même si ce n'est pas nécessairement une solution optimale.

Remarque. Tester la validité d'*un* N-uplet de positions quelconques pour les robots est peu coûteux – quelques multiplications seulement. En revanche, tester les N-uplets valides parmi tous ceux possibles a une complexité polynomiale en le nombre n^r de positions possibles par robot et exponentielle en le nombre N de robots, avec une complexité en $O\big((n^r)^N\big)$. L'approche reste néanmoins valable pour de petites équipes de robots. On peut également se contenter d'un sous-ensemble – échantillonné par un processus de notre choix – car il suffit de trouver un seul N-uplet valide pour que la méthode fonctionne.

6.6.5 *Patrouilles cycliques et considérations sur le long terme*

La définition même de la mission de patrouille décrit le problème comme étant cyclique, au sens où il n'a pas de fin : parcourir une fois la zone à patrouiller ne mène qu'à recommencer, encore et encore jusqu'à l'apparition d'un aléa (détection d'une cible, panne, ordre d'arrêt de la mission, *etc.*). La différence de notre approche réside dans le fait de découper ces différents cycles et de les aborder chacun séparément dans ce que nous appelons une instance du problème. Là où la plupart des algorithmes cherchent donc un schéma de patrouille définitif, régulier ou non, nous proposons quelque chose de plus réactif, chaque nouvelle instance et étape de planification pouvant incorporer les éventuels changements observés, tels une évolution de l'environnement.

Remarque. Il est important de bien distinguer l'aspect cyclique du problème de patrouille et de sa résolution, de l'aspect cyclique des solutions. Le premier est une caractéristique spécifique du problème alors que le second est un choix de conception. Ainsi, nous ne désirons pas ici des solutions cycliques – de type cycles hamiltoniens – car des trajectoires en boucle fermée sont par nature prédictibles et donc préjudiciables dans le contexte antagoniste où nous nous plaçons. Nous préférons élaborer des trajectoires sans schéma reconnaissable, sans boucle ou « cycle » bien qu'elles soient calculées de manière cyclique.

Les sections 6.1 à 6.5 se sont concentrées sur la séparation de la mission en « instances » reprenant l'idée de cycle, et sur la résolution de ces instances. Nous avons alors laissé provisoirement de côté l'aspect global du problème, sur lequel nous souhaitons revenir ici. Se posent en particulier les questions suivantes : comment se combinent les différents plans élaborés, et quel est le résultat global sur la mission ?

Il y a en effet deux aspects important à évaluer. D'une part, sachant que le parcours optimal d'une zone ne définit pas nécessairement un chemin de patrouille optimal comme illustré figure 9 (p. 32), la concaténation de nos différentes plans de patrouille est-elle performante sur la durée ? En d'autres

termes, est-ce que les échantillonnages successifs et leur résolution sont pertinents sur la durée ? D'autre part, la résolution optimale de différentes instances échantillonnées permet-elle d'élaborer des schémas de patrouille stochastiques, c'est-à-dire non-prédictibles ? En d'autres termes, quel est le taux de corrélation entre les différents chemins issus de chaque instance ?

Nous évaluons ces deux aspects – optimalité et non prédictibilité – dans les chapitres 7 et 8. En particulier, nous comparons les différentes formulations IP et discutons l'impact de la durée T des cycles de patrouille. Par ailleurs, nous vérifions l'hypothèse $\mathfrak{u} \approx cte$ sur T.

7 RÉSULTATS EXPÉRIMENTAUX

Tout marche, et le hasard corrige le hasard.
De là vient l'équilibre, et toujours l'ordre éclate.

— Victor Hugo

Ce chapitre présente des résultats expérimentaux issus de la mise en application des considérations théoriques présentées aux chapitres 5 et 6. Il présente les résultats de nos schémas de patrouille, élaborés à partir de différentes formulation IP. Dans un premier temps (section 7.1), nous expliquons comment nous avons intégré notre formalisme et quels choix nous avons faits, nous menant aux résultats présentés par la suite. La section 7.2 présente les résultats préliminaires ayant guidé la suite de nos analyses expérimentales. Les sections 7.3, 7.4 et 7.5 présentent respectivement une analyse des schémas de patrouille calculés, une discussion des aspects liés à l'échantillonnage, et une analyse des performances sur le long terme. Le chapitre se conclut en discutant de l'impact des communications sur les performances.

7.1 INTÉGRATION

Cette section présente certains détails d'intégration du formalisme des chapitres 5 et 6 permettant de comprendre et de reproduire* les résultats expérimentaux présentés dans la suite du chapitre. L'intégration du formalisme et des algorithmes précédemment présentés rassemble deux aspects distincts : l'intégration des modèles d'une part, et le choix et l'utilisation d'un solveur IP d'autre part.

Le code source exploité aux chapitres 7 et 8 est sous licence BSD `https://github.com/cyrobin/patrolling`.

7.1.1 *Intégration des modèles*

Nous détaillons ici l'intégration des modèles utilisés par nos algorithmes de planification de patrouille, pour laquelle nous avons mis en œuvre nos idées développées au chapitre 5. À l'heure actuelle, nos algorithmes de planification de patrouille n'utilisent pas directement la librairie *Gladys* mais sont construits autour d'un équivalent écrit en Python et exploitant les mêmes données de base, et en particulier les cartes au format GeoTIFF exploitant GDAL et les modèles des robots et de mission au format json – voir aussi l'annexe A à ce sujet. Cela nous a permis un premier prototypage rapide dont nous montrons les résultats ici et qui est tout à fait compatible avec l'exploitation future de la librairie *Gladys*.

En particulier, on exploite les données d'entrée suivantes :
— descriptif de la mission (json) ;
— descriptif de chaque robot impliqué dans la mission (json) ;
— cartes d'accessibilité, une par robot (GeoTIFF) ;
— cartes de visibilité, une par robot (GeoTIFF) ;
— carte d'utilité, une pour la mission (GeoTIFF), décrivant l'utilité initiale de chaque position objectif et son importance, et représentant son évolution au cours du temps, entre deux instances.

Le descriptif de la mission précise notamment les fichiers utilisés (dont les cartes), les positions de départ des protagonistes et les paramètres de la mission – ici la durée T d'un cycle). Chaque robot utilise un modèle de déplacement basé sur sa carte d'accessibilité, sa taille et sa vitesse nominale, un modèle de visibilité basé sur sa carte de visibilité, la hauteur de son capteur, le type de capteur, sa portée et sa qualité – les paramètres sont utilisés pour définir la fonction de perception décrite plus loin – et un modèle de communication fonction de la distance. Ces modèles sont facilement extensibles.

Actuellement, nous considérons des robots ayant un angle de vue à 360° et nous ne prenons pas en compte leur orientation. Cela peut s'avérer problématique dans certains cas – voir à ce sujet le chapitre 9 – mais reste suffisant pour nos algorithmes de patrouille et les résultats présentés ici. Ce n'est pas une perte de généralité, mais il est important de considérer que la prise en comte de l'orientation du robot impacte fortement l'étape d'échantillonnage et la complexité du problème instancié.

Par ailleurs, P^r et Q sont basés sur des ensembles de positions 2D, disposées selon une grille cartésienne – des *rasters* GDAL au format GeoTIFF. Ces grilles sont géoréférencées, et différent d'un robot à l'autre – par la taille du robot et par ses capacités d'accès – comme illustré à la figure 32.

À partir du tableau 1, explicitons les fonctions de base utilisées ici :

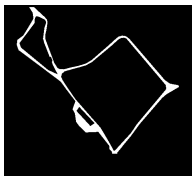
PERCEPTION

$$\phi^r_{pq} = \begin{cases} 1 & \text{si } dist(p,q) = 0 \\ 0 & \text{si } dist(p,q) > sensor\_range \\ \min\left(1, \frac{coef}{g(dist(p,q))}\right) & \text{sinon} \end{cases} \quad (38)$$

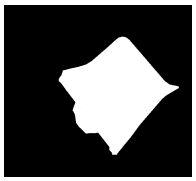
où $dist(p,q)$ est la distance euclidienne entre p et q, $sensor\_range$ la portée maximale du capteur, $coef$ sa qualité, et où g est une fonction log, linéaire, quadratique ou racine décrivant la dépréciation de la perception avec la distance, qui dépend du type de capteur.

MOUVEMENT $c^r_{p_1p_2}$ est une estimation du coût du trajet calculée en prenant la distance du chemin estimé entre p_1 et p_2 pour r, et divisée par la vitesse de r pour obtenir son coût estimé en temps.

(a) Vue satellite



(b) P^{AGV}



(c) P^{AAV}

FIGURE 32 – Exemples de cartes – grilles – GeoTiff utilisées comme modèles d'environnement pour l'accessibilité des robots, ici sur le site de Caylus : (a) présente la vue satellite du site, (b) et (c) sont les grilles respectives représentant l'accessibilité pour les AGVs et AAVs. Les premiers sont restreints aux routes tandis que les seconds sont restreints au champ formant une zone de vol autorisé. Les zones dont l'utilité est significative correspondent à la réunion de P^{AGV} et P^{AAV}.

COMMUNICATION

$$\alpha^{r}_{p_1 p_2} = \begin{cases} 1 & \text{si } dist(p_1, p_2) \leqslant com\_range \\ 0 & \text{sinon} \end{cases} \tag{39}$$

où $dist(p_1, q_2)$ est la distance euclidienne entre p_1 et p_2, et com_range la portée maximale de communication de r. Le modèle est donc actuellement binaire.

UTILITÉ On rappelle que l'utilité est considérée comme constante sur une phase de patrouille, c'est-à-dire lors de l'instanciation par échantillonnage et lors de la résolution de l'instance. Entre deux instances, l'utilité est mise à jour selon le modèle suivant, prenant en compte la durée du plan et les observations résultant de ce plan :

$$u(q, t_0 + T) = u(q, t_0) * \left(1 - \max_{r \in R, p \in \mathcal{P}^r} \phi^{r}_{pq}\right) + g^{u}(q) * T \tag{40}$$

où $\mathcal{P}^r$ est le plan élaboré par le robot r, c'est-à-dire les points $p \in P^r$ par lesquels il passe effectivement, et $g^u(q)$ le taux de croissance de u en q indiquant l'importance du point – plus $g^u(q)$ est élevé et plus il est utile d'observer q souvent. Ainsi, l'utilité restante à la fin d'une instance est toujours une majoration de l'utilité « réelle » au sens de l'oisiveté qu'elle représente : en effet, tout se passe comme si toutes les positions observées

l'étaient au tout début de la phase de patrouille alors que les observations sont en fait réparties sur toute la durée de cette phase.

ÉCHANTILLONNAGE f est une simple fonction de normalisation permettant de transformer l'utilité cumulée ν en la PDF η associée.

Conformément à nos recommandations du chapitre 5, nous avons clairement séparé les algorithmes des modèles sous-jacents. Les fonctions actuellement exploitées sont des fonctions basiques avant tout utilisées pour créer un premier prototype permettant d'évaluer rapidement la viabilité de nos algorithmes. Il est facile de substituer à ces fonctions d'autres fonctions de calculs plus élaborées (lancer de rayons pour la visibilité et la communication, A-star pour les déplacements, *etc.*), par exemple en liant nos algorithmes à la librairie *Gladys*. Le résultat est transparent pour la planification, conformément à notre principe d'API hiérarchique. Cette intégration fait partie de nos priorités à court terme. En attendant, la suite de ce chapitre tout comme le chapitre 9 utilisent les fonctions de prototypes, émulant les résultats de *Gladys*.

7.1.2 *Solveur IP*

Nous avons utilisé le solveur IP GLPK. Nous détaillons ce choix et le fonctionnement de ce solveur dans l'annexe B. Plus spécifiquement, la section 7.2 utilise directement les binaires de la librairie (lp_solve) tandis que les autres résultats expérimentaux ont été obtenus à travers la librairie PyMathProg* fournissant une API Python à GLPK à travers PyGLPK. Cette API a été retenue pour sa simplicité et sa lisibilité. En outre, il est très important de noter que GLPK est un solveur *déterministe*. Cela permet de garantir la reproductibilité des résultats pour une instance donnée – l'échantillonnage restant lui stochastique.

À propos de PyMathProg, voir http://pymprog.sf.net/

Dans le présent chapitre, tous les résultats sont issus d'une résolution centralisée : le solveur utilise l'ensemble des données disponibles pour élaborer en même temps les plans de chaque robot, ce qui permet une coordination optimale entre les robots, au contraire du chapitre 8 qui étudie une approche séquentielle et décentralisée du problème.

Le solveur GLPK, de par son fonctionnement, est capable d'évaluer sa résolution du problème IP qui lui est donné. En particulier, il est capable d'indiquer si des solutions existent ou non, et lorsqu'il en a renvoyé une il indique si elle est optimale ou non. Si la solution est sous-optimale, il peut fournir des bornes sur son optimalité, bien que celles-ci se révèlent assez larges en pratique. En outre, il est possible de fixer une limite de temps au bout de laquelle le solveur s'arrête et renvoie la meilleure solution trouvée jusque-là, généralement sous-optimale. Nous faisons un usage systématique de cette fonction, avec une limitation du temps de calcul à cinq minutes, ce qui pour nous correspond à un « maximum acceptable* » en cours de mission pour planifier le prochain chemin.

Il est courant de laisser tourner un solveur IP pendant plusieurs heures pour évaluer ses capacités et obtenir le meilleur résultat possible ; nous trouvons ces temps de calculs trop longs pour notre application qui vise la replanification en cours de mission, c'est pourquoi nous nous limitons à quelques minutes de calculs seulement.

Il n'est pas facile d'évaluer *a priori* quelles sont les capacités du solveur face à un nouveau problème IP car ses performances dépendent du nombre de variables et de contraintes mais aussi de sa capacité à exploiter les liens entre elles, c'est-à-dire en quelque sorte de la « topologie » du problème. Or, si nos problèmes TOP et SP sont proches du classique TSP, ils sont différents dans leur complexité – plus grande – et dans les variables – il y a des variables et des relations supplémentaires. Afin d'évaluer grossièrement les capacités de notre solveur, nous avons dans un premier temps lancé de nombreuses instances afin de définir quels étaient les jeux de paramètres acceptables pour le solveur. Ceci est présenté dans la section 7.2 ; les sections suivantes évaluent alors plus en détails les performances de notre approche dans ces bornes.

7.2 RÉSULTATS PRÉLIMINAIRES

Dans cette section, nous essayons de délimiter le domaine de fonctionnement de notre solveur IP, c'est-à-dire de déterminer les jeux de paramètres acceptables permettant de renvoyer une solution à l'instance, si possible optimale. Dans un premier temps, nous évaluons la complexité des deux types de formulations TOP et SP pour le nombre de variables et de contraintes – section 7.2.1 – avant d'évaluer la performance de notre solveur sur une échelle de complexité – section 7.2.2.

7.2.1 *Complexité des formulations IP*

La complexité d'une formulation IP est difficile à évaluer, mais il est commun d'indiquer le nombre de variables et le nombre de contraintes du problème. Nous les indiquons ci-dessous en fonction des différents paramètres d'entrée :

- N le nombre de robot – le cardinal de R.
- n^r le nombre de positions accessibles au robot r – le cardinal de P^r. Par commodité, nous considérons que $\forall r \in R : n^r = n$.
- b^r le facteur de branchement du graphe d'accessibilité $\mathcal{G}^r$ de r. Le nombre d'arcs de ce graphe est donc $n^r b^r$. Par commodité, nous considérons que $\forall r \in R : b^r = b$.
- m le nombre de positions observables – le cardinal de Q.

FORMULATION TOP Les formulations TOP-pos – section 6.5.1 – et TOP-perc – section 6.5.2 – proposent deux formulations IP presque identiques, ayant le même nombre de variables* et de contraintes. Nous calculons cette complexité ci-dessous : le nombre de variables est de

En grande majorité, les variables des formulations TOP et SP sont binaires.

$$O(Nnb) = \underbrace{O(Nn)}_{(14)} + \underbrace{O(Nnb)}_{(15)} + \underbrace{O(Nn)}_{(16,17)} + \underbrace{0}_{(18)} + \underbrace{O(Nn)}_{(29)} + \underbrace{0}_{(30)}$$

et le nombre de contraintes de

$$O(Nnb) = \underbrace{0}_{(14)} + \underbrace{O(Nn)}_{(15)} + \underbrace{O(Nn)}_{(16,17)} + \underbrace{O(N)}_{(18)} + \underbrace{O(Nnb)}_{(29)} + \underbrace{O(Nn)}_{(30)}$$

En outre, la linéarisation de (15) en (19) et (20) ne change pas l'ordre de complexité : cela double le nombre de contraintes associées à l'équation, le nombre de variables restant inchangé. En revanche, on remarque à travers les équations (29) et (30) que les SEC MTZ ont un impact significatif, en particulier sur le nombre de contraintes comme cela avait été avancé à la section 6.5.4.

FORMULATION SP La section 6.5.3 propose une formulation de type SP différant largement des formulations TOP. Le nombre de variables associées est de

$$O(m + Nnb) = \underbrace{O(m)}_{(22)} + \underbrace{O(Nnb)}_{(23)} + \underbrace{O(Nn)}_{(24,25)} + \underbrace{0}_{(26)} + \underbrace{O(Nn)}_{(29)} + \underbrace{0}_{(30)}$$

et celui de contraintes de

$$O(m + Nnb) = \underbrace{0}_{(22)} + \underbrace{O(m)}_{(23)} + \underbrace{O(Nn)}_{(24,25)} + \underbrace{O(N)}_{(26)} + \underbrace{O(Nnb)}_{(29)} + \underbrace{O(Nn)}_{(30)}$$

En outre, la linéarisation de (23) en (27) et (28) impacte sensiblement la complexité, le nombre de contraintes associées passant de $O(m)$ à $O(Nnm)$ – le nombre de variables est inchangé. La complexité résultante devient alors $O(Nn(m + b))$. De manière similaire aux formulations TOP, les SEC MTZ ont ici aussi un impact significatif sur le nombre de contraintes.

IP	variables	contraintes
TOP	$O(Nnb)$	$O(Nnb)$
SP	$O(m + Nnb)$	$O(Nn(b + m))$

TABLE 3 – Récapitulatif du nombre de variables et de contraintes des formulations ILP utilisées. On remarque que les SP sont plus complexes que les TOP. Par ailleurs les variables sont principalement binaires.

Nous résumons les contraintes et variables des formulations linéarisées dans le tableau 3. En comparant l'ordre de grandeur usuel des variables, on peut raisonnablement s'appuyer sur les relations $N \ll n \sim m$. Le facteur de branchement b des graphes d'accessibilité varie lui grandement selon la structure de graphe. À ce titre, il est avantageux d'éviter de travailler sur des graphes complets pour lesquels $b \sim n$, ce qui résulte en une complexité quadratique en le nombre de positions accessibles échantillonnées pouvant

vite devenir inabordable. On préférera un facteur de branchement faible tel que $N \sim b \ll n \sim m$.

Parmi les quatre paramètres définissant le nombre de variables et de contraintes du problème, et par là même indiquant la complexité de la formulation IP, il est important de noter que la valeur de N est imposée par la mission, tandis que n et m sont choisis lors de l'étape d'échantillonnage. b est lui choisi à la construction des graphes $\mathcal{G}^r$. L'influence de n et m se ressent sur la proximité du problème instancié avec le problème de départ, et l'on souhaiter éviter des valeurs trop faibles car elles sont aussi garantes de l'étendue de l'espace de recherche des solutions. Le facteur de branchement b a quant à lui pour principal effet de contraindre les mouvements des robots : une faible valeur de b poussera à aller de proche en proche entre les positions accessibles, alors qu'une grande valeur de b comme dans un graphe complet donne une grande liberté dans les choix des déplacements successifs. À ce titre, on peut s'attendre à ce que b ait une influence moins significative que n et m sur la qualité des solutions vis-à-vis du problème de patrouille.

La section suivante prolonge sur cette analyse et présente les performances – le taux de succès et le taux d'optimalité – du solveur IP pour différents jeux de paramètres.

7.2.2 *Performances face à la complexité*

Après avoir explicité la complexité théorique des problèmes TOP et SP en termes de nombres de variables et de contraintes, nous cherchons ici à quantifier les limites sur les valeurs de N, b et n permettant de fixer un cadre à l'analyse plus en profondeur de nos algorithmes. La figure 33 présente le taux de réussite de notre solveur GLPK face à des problèmes TOP de complexité croissante, chaque problème étant une instance de patrouille avec un jeu de paramètres donné. Par « réussite » nous entendons « le solveur a renvoyé une solution valide » : la pertinence des solutions n'est pas prise en compte ici et sera évoluée spécifiquement dans les sections suivantes. L'abscisse de la figure est ainsi : *grid* représente la complexité d'un environnement en forme topologie de grille en 4-connexité dans lequel on a un facteur de branchement moyen $b \leqslant 4$ et un nombre de nœuds $n \in [14-20]$. Les autres graduations de l'abscisse correspondent à la valeur de bn avec $b = n$ (par exemple $b = n = 6$ et $bn = 6^2$), ce qui signifie que les graphes d'accessibilité sont des graphes complets*.

En théorie, $b = (n-1)$ pour un graphe complet, mais nous rajoutons ici un lien vers les positions de départs qui ne sont pas comptées dans n.

Chaque courbe correspond à un nombre N de robots différent et le taux de réussite affiché est une moyenne sur plusieurs dizaines de tests. On constate une chute progressive du taux de réussite en fonction de la complexité, avec une « cassure » pour bn entre 10^2 et 12^2 qui s'explique par la limite de temps de calcul imposé, d'environ cinq minutes ici : avant cette cassure, le solveur dispose d'assez de temps pour trouver au moins une solution valide et l'améliorer avec le temps – voir l'annexe B sur le fonctionnement de GLPK. Après la cassure, le temps disponible est insuffisant

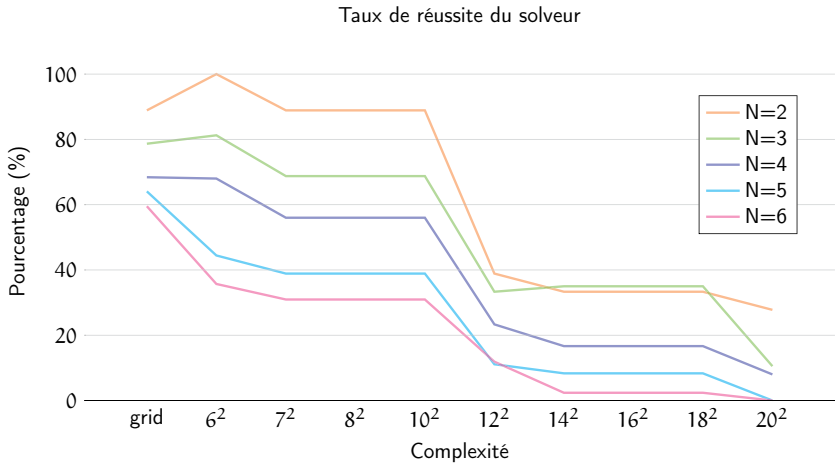


FIGURE 33 – Taux de réussite du solveur GLPK face à des problèmes TOP de difficulté croissante. Les performances sont globalement décroissantes avec la complexité, et l'on remarque une cassure nette dans cette évolution, s'expliquant par la limite de temps de calcul imposée au solveur et atteinte dans le cas des problèmes plus complexes.

pour trouver une solution en nombres entiers respectant les contraintes. On constate par ailleurs que la complexité *grid* est parmi la complexité la plus simple : cela s'explique par le faible facteur de branchement.

En complément du taux de réussite, la figure 34 indique le taux d'optimalité du solveur sur les problèmes TOP, c'est-à-dire le pourcentage de solutions optimales *parmi les solutions renvoyées*. C'est un indicateur supplémentaire des capacités et limites du solveur : la qualité des solutions renvoyées ne pouvant qu'augmenter avec le temps alloué au solveur, plus le taux d'optimalité est important et plus le solveur est confortable face au problème présenté.

Les résultats au-delà de 10^2 sont indiqués en pointillés car ils ne sont pas considérés comme significatifs à cause d'un trop petit nombre d'exemples – ce nombre est lié au taux de réussite très faible. Pour les autres, deux constats peuvent être faits : d'une part, lorsqu'une solution est renvoyée, elle a une forte chance d'être optimale. Cela signifie que le solveur semble converger assez vite vers une solution optimale une fois qu'une solution valide a été trouvée. D'autre part, il semble que les problèmes plus difficiles, c'est-à-dire avec un taux de réussite plus bas, suivent une règle de « tout ou rien » signifiant qu'une solution est rarement trouvée mais que celle-ci a tendance à être optimale ou à permettre de rapidement converger vers l'optimal lorsque c'est le cas.

Remarque. La figure 34 indique que dans l'ensemble les solutions aux problèmes instanciés renvoyées par GLPK semblent bonnes, une grande partie étant optimales, mais cette optimalité est calculée par rapport à l'instance : elle n'indique rien de la pertinence des solutions par rapport au problème d'origine, c'est-à-dire avant l'étape d'instanciation par échantillonnage. *A fortiori*, elle n'indique rien de la pertinence des solutions dans leur enchaî-

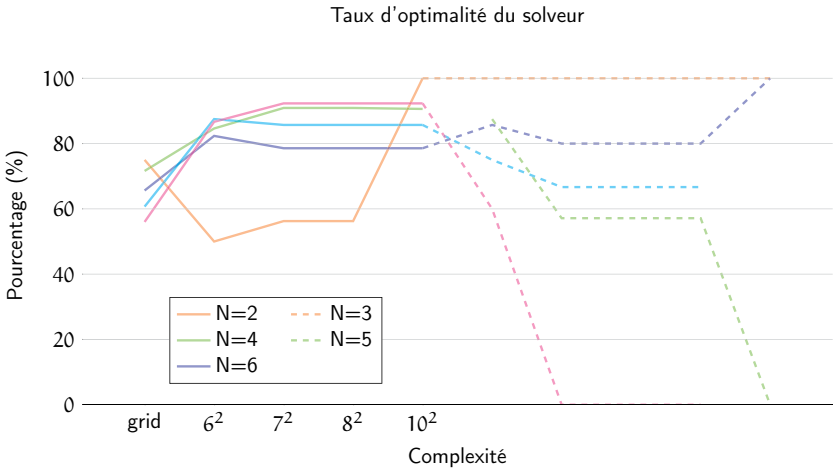


FIGURE 34 – Taux d'optimalité du solveur GLPK face à des problèmes TOP de difficulté croissante, lorsqu'une solution valide est effectivement renvoyée. Ce taux d'optimalité est globalement important pour des problèmes simples – pour les autres, les données ne sont pas significatives.

nement au cours du temps.

Les figures précédentes indiquent des marges de manœuvre acceptables au niveau des paramètres pour des instances de type TOP ; de la même manière, la figure 35 indique le taux de réussite du solveur GLPK pour différents jeux de paramètres pour des instances de type SP.

Plus spécifiquement, les paramètres n et b sont repris de la complexité *grid* précédente, et l'on étudie alors les taux de réussite pour différentes valeurs de N et de m, le nombre de positions observées. On voit que ce paramètre influe beaucoup sur la performance du solveur comme cela était attendu suite à notre analyse théorique. Le taux de réussite du solveur est une fonction globalement décroissante de N et m. Néanmoins, on remarque que même pour des très faibles valeurs de m et de N, le solveur affiche des résultats médiocres. On peut donc en conclure que cette formulation, bien qu'attrayante en théorie, s'avère problématique en pratique. En particulier, elle contraint énormément les valeurs des paramètres, dont n et b. On peut contenir la complexité en restreignant drastiquement n – par exemple $n < 10$ – de manière à obtenir des taux de réussite proches de 100% même avec une valeur de m élevée. Cela s'explique en constatant que la complexité du nombre de contraintes est en $O(Nn(b+m))$, soit en $O(Nnm)$ en constatant que $b \ll m$ dans la grande majorité des cas.

Remarque. Dans le cas de la formulation SP, il est important de noter l'influence d'un paramètre caché : la fonction ϕ, et plus exactement sa portée, c'est-à-dire la portée des capteurs. En effet, nous avons remarqué que lorsque la portée augmente, c'est-à-dire lorsqu'il existe de nombreux liens de visibilité entre les positions $p \in P^r$ des robots et les positions observables $q \in Q$, alors le solveur arrive plus facilement à trouver une solution : cela

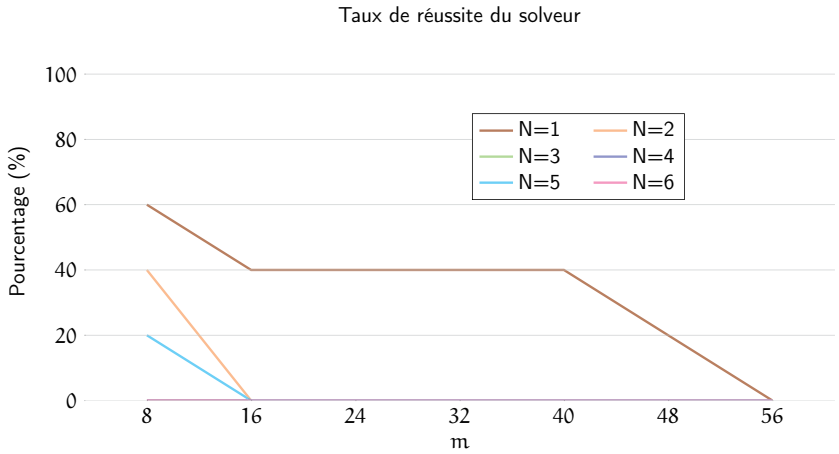


FIGURE 35 – Taux de réussite du solveur GLPK face à des problèmes SP de difficulté croissante. Les performances sont globalement mauvaises, et décroissent vite vers zéro dans les problèmes multirobots ($N \geqslant 2$). Ceci suggère que la complexité des problème SP est très importante, comme attendu. En conséquence, il est nécessaire de limiter drastiquement les valeurs des paramètres afin de remonter le taux de réussite à des valeurs acceptables pour une utilisation pratique.

se comprend par le fait que presque n'importe quelle position p accessible gagne en intérêt. Cela n'améliore pas la qualité des solutions pour autant.

En résumé, cette analyse préliminaire nous indique quels sont les paramètres influençant les performances du solveur concerntant le renvoi d'une solution et l'optimalité de la solution renvoyée. Elle nous indique quantitativement quels jeux de paramètres sont acceptables, ce qui va guider notre analyse en détails menée dans les sections suivantes. En outre, on retient qualitativement les résultats suivants :

— Afin de gérer la complexité, il est important de limiter le facteur de branchement des graphes d'accessibilité ; vu que nos robots se déplacent de proche en proche, cela n'est pas vraiment contraignant.
— La formulation TOP renvoie souvent une solution au problème IP, qui a de plus des chances significatives d'être optimale.
— En revanche la formulation SP est très complexe et termine seulement pour des jeux de paramètres très contraints ; les solutions obtenues sont très rarement optimales.

Remarque. Il est important de se rappeler que les limites identifiées ici sont valables pour le solveur GLPK, qui n'est *pas* le solveur le plus performant existant : les résultats sont donc à prendre comme une limite basse des possibilités des formulations proposées.

Après avoir déterminé les jeux de paramètres acceptables par le solveur IP, nous pouvons maintenant analyser la qualité des solutions au-delà du simple cadre de l'instance. En effet, l'instance IP à résoudre n'est qu'un problème artificiel, créé dans le but de nous aider à résoudre le problème original de patrouille. Nous avons étudié nos algorithmes dans quatre environnements distincts, deux d'entre eux étant artificiels et deux autres tirés d'environnements existants. Leurs propriétés sont résumées au tableau 4.

Environnement	Type	Réel / Artificiel	Structuré	Taille
Parking du LAAS	Urbain	Réel	Oui	500*700
Caylus	Naturel	Réel	Oui	550*500
Manhattan	Urbain	Artificiel	Oui	591*845
Plaine	Naturel	Artificiel	Non	591*845

TABLE 4 – Liste et propriétés des environnements de tests utilisés pour la validation expérimentale. La quatrième colonne indique la présence d'éléments structurant l'environnement (obstacles) ; la dernière précise la taille des grilles en nombre de cellules. Cette dernière donnée est à pondérer par le nombre de cellules effectivement actives : il peut y avoir de nombreux obstacles, comme pour le LAAS et Caylus.

La suite de cette section présente visuellement ces environnements et montre des exemples de solutions permettant d'évaluer la pertinence des solutions. Les plans de patrouille élaborés et montrés ici correspondent exclusivement à un seul cycle de planification, c'est-à-dire à un même cycle d'échantillonnage et de résolution du problème IP.

7.3.1 *Environnement « Parking du LAAS »*

L'environnement « parking du LAAS » est tiré d'un environnement d'expérimentation réel dont la figure 36 montre une vue satellite et les grilles utilisées comme carte d'accessibilité.

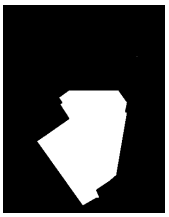
Par la suite, nous illustrons les plans de patrouille sur des cartes-grilles similaires à celles utilisées pour P^r mais exprimant l'utilité des zones associées plutôt que l'accessibilité des robots, comme à la figure 37. Cette figure est importante car elle introduit les notations et la légende qui seront utilisées de manière récurrente pour illustrer les performances des algorithmes de planification de patrouille. Ainsi, on y voit les éléments suivants :

La LUT d'utilité étant dynamique, il n'est pas possible de comparer différentes cartes d'utilité sur des critères d'intensité de couleur uniquement.

— la carte d'utilité, sur une échelle à une dimension en noir et blanc, dont la LUT indiquée sur le côté est dynamique* – elle sera donc précisée avec chaque carte d'utilité. Très souvent, les zones noires, correspondant à une utilité nulle, peuvent être associées à des obstacles, notamment pour les AGVs.

(a) Vue satellite

(b) $\mathrm{P}^{\mathrm{AGV}}$



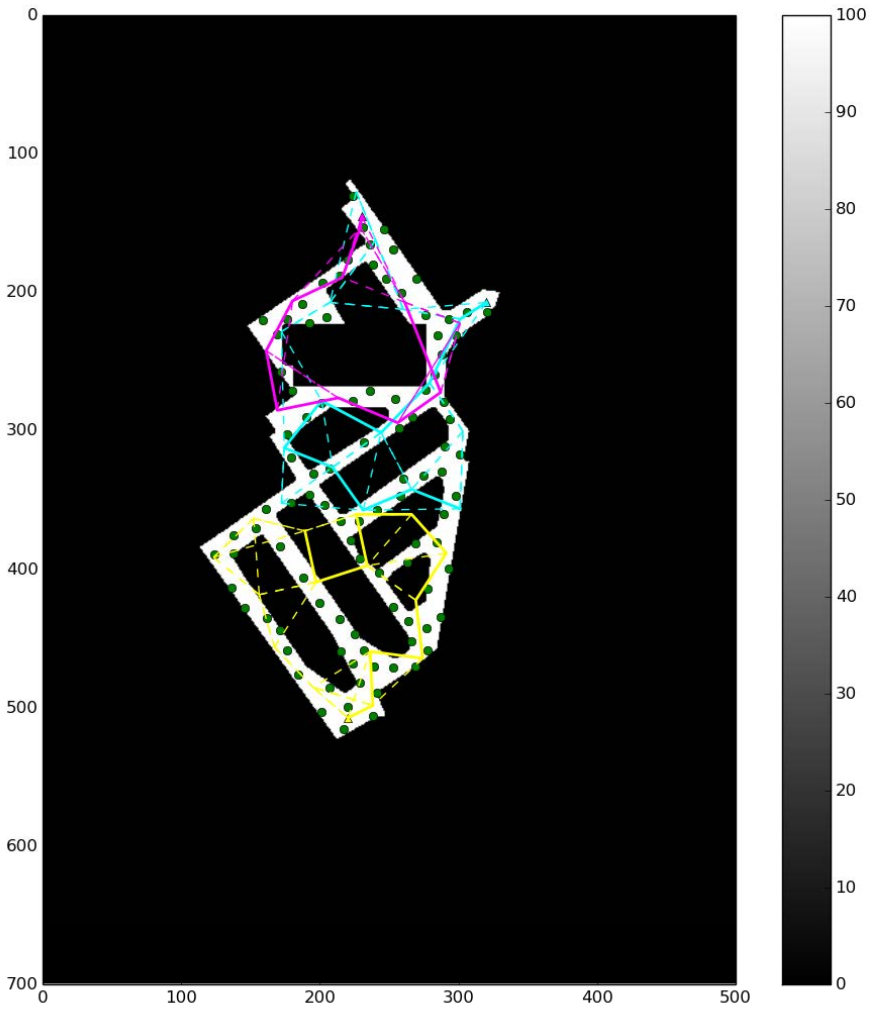
(c) $\mathrm{P}^{\mathrm{AAV}}$

FIGURE 36 – Exemples de cartes (grilles) GeoTiff utilisées comme modèles d'environnement pour l'accessibilité des robots sur le site du parking du LAAS : (a) présente une vue satellite du site, (b) et (c) sont les grilles respectives représentant l'accessibilité pour les AGVs et AAVs. Les premiers sont restreints par la présence des voitures tandis que les seconds n'ont pas l'autorisation de survoler les bâtiments. Q, l'ensemble des positions objectifs observables, est similaire à $\mathrm{P}^{\mathrm{AGV}}$.

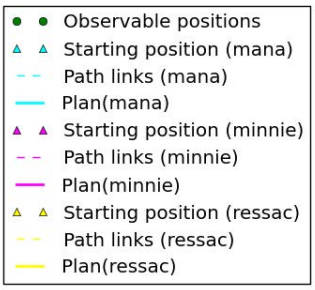
— les points observables sont affichés en vert : ils sont tirés de l'étape d'échantillonnage.
— $\mathcal{G}^{\mathrm{r}}$ est indiqué pour chaque robot (nommés *mana*, *minnie* et *ressac*) en pointillés : les points d'intersection sont les sommets du graphe, et les segments sont les arcs non orientés. On rappelle que nous ne considérons pas ici l'orientation des robots. *mana* et *minnie* sont des AGVs, *ressac* un AAV.
— Les triangles colorés désignent les positions initiales des robots.
— Les chemins colorés en ligne continus indiquent les plans des robots.

La figure 37 illustre l'utilité au début de la mission : hormis les obstacles aux AGVs qui ont une utilité associée nulle, les autres positions ont une utilité uniforme, d'une valeur arbitraire égale à 100. Celle-ci évolue à la fin d'un cycle de patrouille comme indiqué par la formule (40) (p.121). La figure 38a montre l'utilité en fin de cycle associée aux plans calculés en formulation TOP à la figure 37. On voit ainsi que l'utilité a bien diminué en moyenne, mais que son maximum global a augmenté sur les zones non perçues par les plans.

Ces plans sont issus de la résolution d'une instance : il va de soi que, conformément à la théorie, ces plans ne sont pas déterministes même en cas de résolution optimale de l'instance, ce qui est le cas ici. En effet, l'étape d'échantillonnage introduit de l'aléatoire dans les solutions calculées, comme illustré à la figure 38b. Ici, les plans correspondent ainsi au résultat



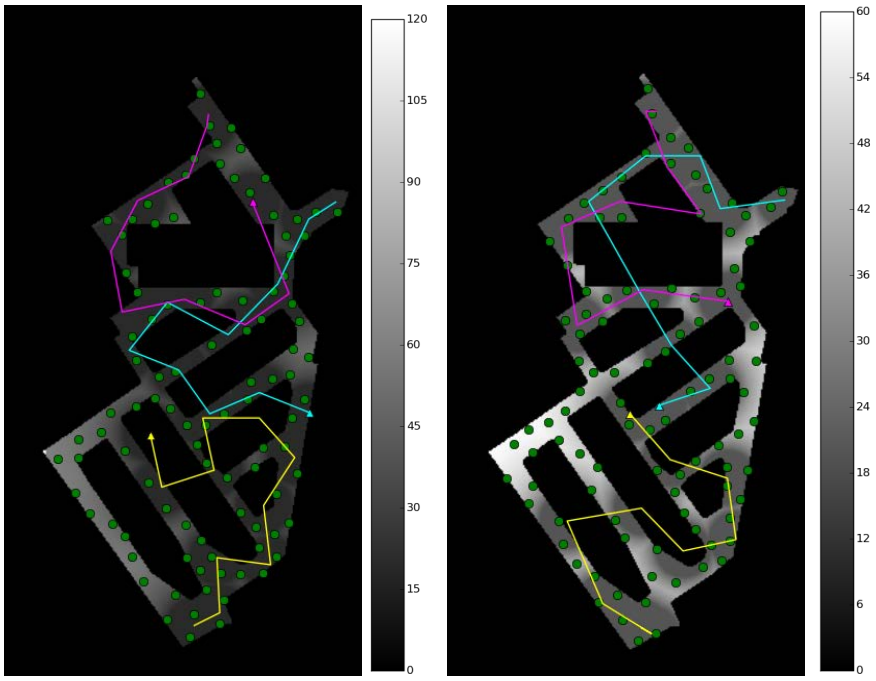
(a) Mission de patrouille



(b) Légende

FIGURE 37 – Illustration d'une mission de patrouille, avec en fond la carte d'utilité et sur le côté la LUT associée. Les positions de départ des robots, leurs graphes d'accessibilité et leurs plans sont indiqués par un code de couleur (une couleur par robot). Les notations et la légende fournies ici sont reprises dans toutes les illustrations de patrouilles. ($N = 3, b = 3, n = 16, m \approx 100$)

de nos algorithmes appelés avec les *mêmes* paramètres, et l'on perçoit les nombreuses différences dans l'échantillonnage et dans les plans résultants. D'ailleurs, cela se ressent également sur la qualité de la solution, avec un maximum d'utilité moitié moindre que pour la première solution. Ce phénomène de stochasticité et les mesures d'utilités associées sont analysés plus en détails à la section 7.5 lors de notre étude des performances des successions de plans, ou autrement dit de la planification sur le long terme.



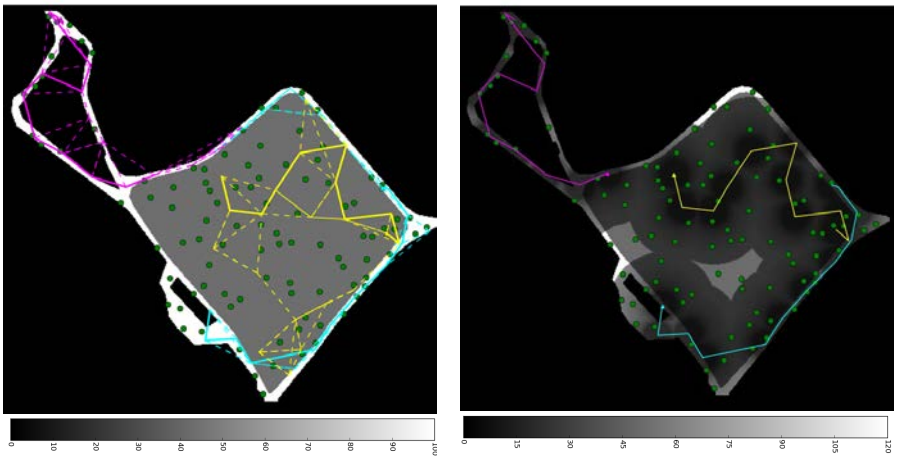
(a) Plan 1 (b) Plan 2

FIGURE 38 – Illustrations de la stochasticité des algorithmes sur l'environnement « parking du LAAS » : à partir de la même configuration initiale les algorithmes peuvent élaborer des jeux de plans différents. ($N = 3, b = 3, n = 16$)

7.3.2 *Environnement « Caylus »*

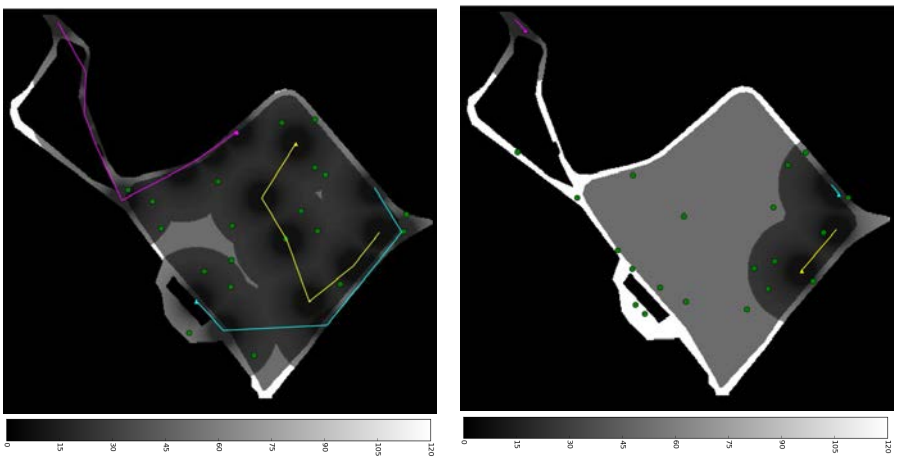
L'environnement « Caylus » est tiré d'un environnement d'expérimentation réel, dont la figure 32 (p.121) montre une vue satellite et les grilles utilisés comme carte d'accessibilité.

Ce type d'environnement est assez différent de l'environnement « parking du LAAS », notamment pour les AGVs : ceux-ci sont en effet assez contraints par un effet « couloir » des routes dues aux chemins qu'ils doivent emprunter. À l'inverse, les AAVs sont ici peu contraints, et l'on peut noter que la densité de l'échantillonnage peut laisser des zones non représentées dans P^{AAV} : ceci est source de stochasticité tout autant que de disparité dans les résultats.



(a) Données initiales et échantillonnage (b) Plan issu d'un TOP

FIGURE 39 – Illustrations du calcul d'un plan par formulation TOP sur l'environnement « Caylus » : on s'aperçoit que la structure de G^r est fortement contrainte pour les AGVs (effet « couloir »), alors que pour de larges zones accessibles (comme c'est le cas pour l'AAV Ressac ici), l'échantillonnage aléatoire a tendance à laisser des zones vides d'échantillons. Ceci peut être source de disparité dans les performances finales. ($N = 3, b = 3, n = 20$)



(a) Plan correct issu d'un SP (b) Plan médiocre issu d'un SP

FIGURE 40 – Deux exemples de plans calculés à partir d'une formulation SP sur l'environnement « Caylus » : on s'aperçoit que la qualité du plan (a) semble très correcte, similaire à celle de la figure 39b, alors que celle du plan (b) est médiocre. Ce dernier est en effet une ébauche de plan, l'algorithme n'ayant pas pu converger vers une meilleure solution dans le temps de calcul imparti. Malheureusement, ce sont ce type de solutions qui sortent le plus souvent. ($N = 3, b = 3, n = 12, m = 20$)

Comparons le plan illustré à la figure 39b, représentatif des plans élaborés sur Caylus à partir d'une formulation TOP, avec d'autres plans issus eux d'une formulation SP, dont la figure 40 montre deux exemples : on remarque ainsi qu'il est possible d'obtenir des plans de qualité similaire – figure 50e – mais aussi des plans de qualité bien moindre, très médiocres – figure 50f. En pratique, ce sont malheureusement surtout ces derniers que l'on obtient, lorsque l'algorithme renvoie une solution valide. En fait, ces plans sont comme « incomplets » : ils résultent d'une convergence incomplète du solveur IP, n'ayant pas eu le temps d'explorer assez l'espace des solutions valides pour renvoyer une solution de meilleure qualité. Typiquement, ces plans n'exploitent que très partiellement le temps T mis à disposition pour les robots. On retrouve donc visuellement ici des résultats dans la continuité de notre analyse préliminaire – section 7.2.2.

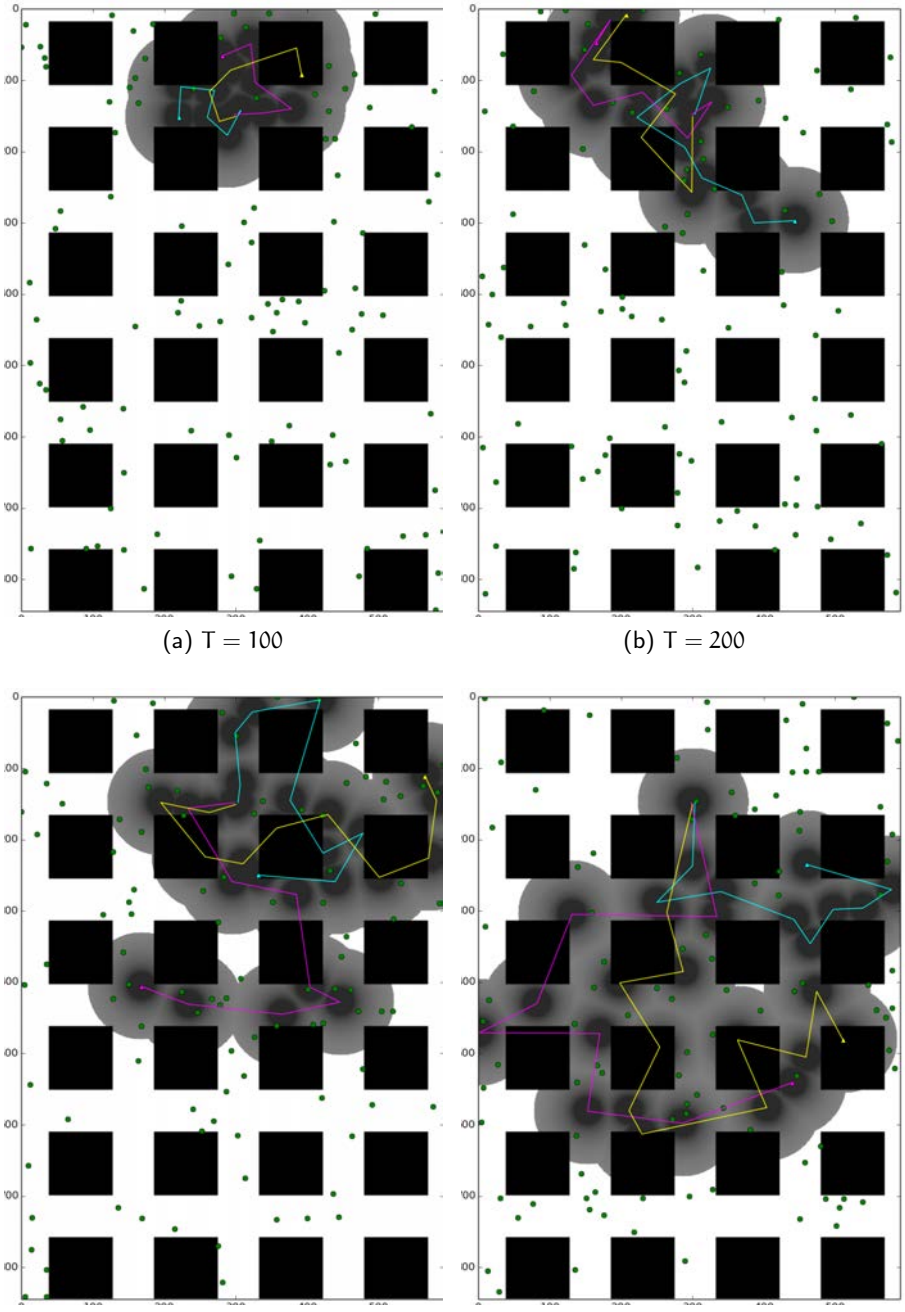
Bien qu'il soit possible de jouer sur le nombre de lancements pour obtenir statistiquement des solutions correctes, cette solution n'est pas envisageable en conditions réelles. En revanche, ces premiers résultats indiquent que des solutions de qualité peuvent émaner des formulations SP, il serait maintenant intéressant d'utiliser un solveur plus performant qui nous permettrait alors de résoudre avec plus de succès les mêmes instances, voire des instances plus complexes encore. C'est une solution durable mais coûteuse en temps de développement, qu'il est néanmoins nécessaire d'intégrer pour passer outre les performances de ce premier prototype.

Enfin, le lecteur attentif aura peut-être remarqué le faible nombre de positions observables échantillonnées. Ces valeurs faibles sont nécessaires pour permettre la convergence de l'algorithme, mais la figure 40 en montre le contrepoint : des zones entières de la carte sont exclues de cet échantillonnage. Ce phénomène impacte plus durement les zones étroites comme la boucle en haut à gauche dans le cas de Caylus.

7.3.3 *Environnement « Manhattan »*

L'environnement « Manhattan » est un environnement artificiel, archétype des environnements de type Manhattan – voir à ce propos la figure 6 p. 25. Bien que ce soit un environnement artificiel, son intérêt est double : d'une part c'est un environnement de test couramment rencontré – son usage facilite donc les comparaisons ; et d'autre part il permet de rendre compte de beaucoup d'environnements réels, comportant des obstacles non reliés entre eux bien que généralement repartis moins symétriquement. Par exemple, sa structure se rapproche de celle de l'environnement « parking du LAAS », en plus large et selon un schéma répété.

La figure 41 compare différents plans obtenus à partir de formulations TOP différant par leur période T indiquant le coût maximal des plans. Les autres paramètres sont identiques, au maximum des capacités acceptables pour le solveur GLPK. On remarque nettement l'impact de la période T sur les différents plans : c'est elle qui définit l'étendue des trajectoires.



(a) T = 100 (b) T = 200

(c) T = 300 (d) T = 500

FIGURE 41 – Planification en formulation TOP pour des périodes de durées variées (les autres paramètres restent inchangés). L'échelle d'utilité est la même pour tous, allant de 0 (noir) à 120 (blanc). ($N = 3, b = 3, n = 20$)

De manière inattendue, le solveur rencontre en fait plus de difficulté à résoudre les problèmes pour une valeur de T élevée, alors que ce paramètre n'apparaît pas directement dans la complexité des problèmes. Nous proposons l'explication suivante : il est assez notable que plus T est élevé est plus les positions échantillonnées sont espacées – elles sont échantillonnées dans l'espace naturellement accessible pour un coût inférieur à T. Hors, la dispersion des positions accessibles mène à un ensemble de solutions possibles très disparates : nous pensons que c'est ce qui gène le solveur dans sa convergence, l'espace des solutions ayant une topologie plus difficile à explorer.

Par ailleurs, cette répartition d'un même nombre d'échantillons sur des espaces de taille variable impacte directement la qualité des solutions trouvables et effectivement trouvées. On retrouve en particulier le phénomène déjà identifié avec les formulations SP et leur contraintes sur le nombre d'échantillons : des « trous » apparaissent dans l'espace d'origine, de sorte que le problème instancié varie beaucoup d'une instance à l'autre et peut être très éloigné du problème d'origine.

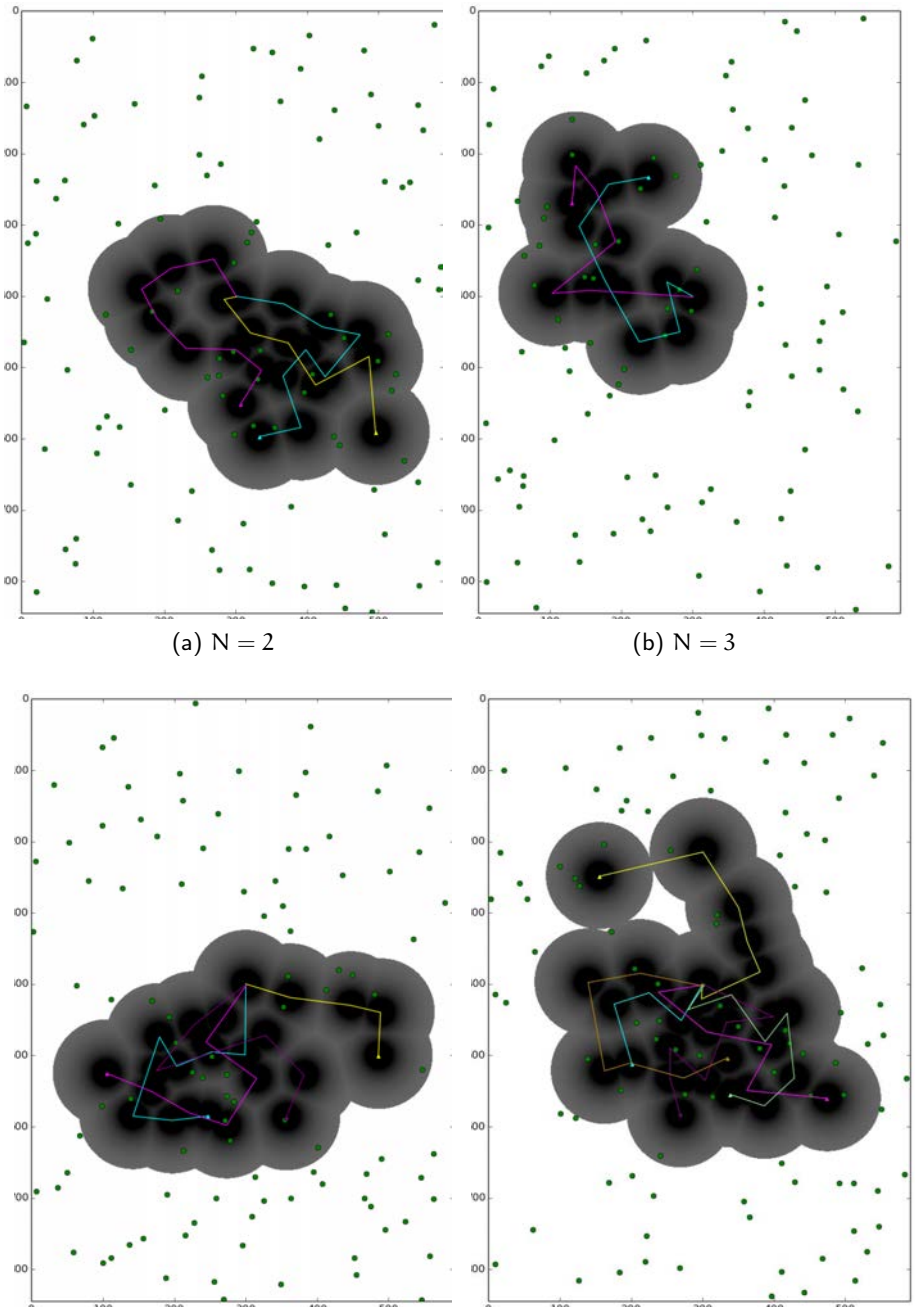
Parfois, T *peut être fixé par des contraintes externes (pour transmettre des rapports réguliers et impératifs à l'opérateur par exemple), mais il y a souvent une certaine marge de manœuvre qu'il faut exploiter au mieux.*

Tout cela nous pousse à fixer les valeurs de T avec soin*, de manière empirique, pour obtenir le meilleur compromis entre une bonne exploration de l'environnement et un sous-échantillonnage problématique. Par ailleurs, on peut aussi se poser la question du meilleur compromis entre « planifier une fois pour une période K.T » ou « planifier K fois successivement pour une période T ». Cet aspect est abordé dans la section 7.5 présentant les performances sur le long terme.

7.3.4 *Environnement « Plaine »*

L'environnement « Plaine » est un environnement artificiel constitué d'un large espace vide d'obstacle. Son intérêt est double également : d'une part il est l'archétype de certaines parties d'environnements voire de certains environnements en entier, notamment pour les AAVs – voir par exemple P^{AAV} pour les environnements « parking du LAAS » et « Caylus ». D'autre part il permet d'étudier le comportement des algorithmes de planification de patrouille en l'absence d'éléments structurants. En effet, il a été évoqué dans l'état de l'art que la structure de l'environnement avait un impact non négligeable sur les performances des algorithmes de planification, et nous souhaitons voir ce qu'il en est pour les nôtres.

La figure 42 présente des plans issus de formulations TOP pour un nombre de robot différents ; les autres paramètres, notamment le nombre de positions accessibles échantillonnées, varient pour exploiter au mieux les capacités du solveur. Il est très remarquable que, quel que soit le nombre de robots impliqués dans la mission, il apparaît de nombreuses redondances entre les trajectoires des robots. Ceci est très problématique car ce phénomène de concentration des robots empêche de tirer pleinement partie de leurs capacités coopératives – on préférerait voir des schémas de déploiement « en



(a) N = 2 (b) N = 3

(c) N = 4 (d) N = 6

FIGURE 42 – Planification en formulation TOP dans un environnement dépourvu d'éléments structurants (obstacles). On remarque une répartition sous-optimale de chemins des robots. L'échelle d'utilité est la même pour tous, allant de 0 (noir) à 120 (blanc). ($b = 3$,n varie inversement à N)

étoile ». En outre, cela peut poser des problèmes locaux de *path planning* lors de la phase d'exécution.

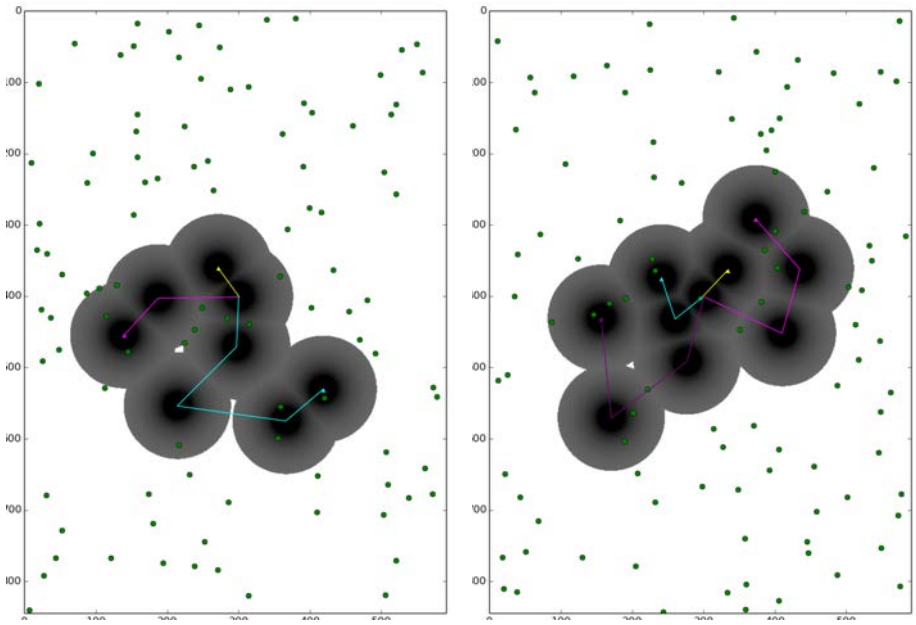
Ce phénomène est néanmoins très compréhensible : il provient conjointement de la phase d'échantillonnage et de la modélisation de la perception lors de la résolution du problème IP. En effet, la phase d'échantillonnage se fait de manière indépendante pour chaque robot – il n'y a pas de pré-attribution d'un secteur à chacun. Par conséquent les positions accessibles échantillonnées pour chaque robot sont potentiellement redondantes. Hors, c'est sur cette base que planifie le solveur, et les formulations TOP ne permettent pas de détecter ces redondances comme cela avait été souligné au chapitre 6. Nous rappelons que cela vient du fait que la perception est prise en compte dès l'étape d'échantillonnage : le solveur est donc « aveugle » face aux liens de perceptions et aux redondances existantes. Pour lui, il n'y a qu'un ensemble de positions accessibles ponctuelles, reliées entre elles par des coûts de déplacements. C'est dans ce type de configuration qu'on voit clairement les limites des hypothèses de simplification qui peuvent être faites lors de la phase de modélisation.

En comparaison, la figure 43 propose des plans calculés à l'aide d'une formulation SP : ce type de formulation prend explicitement en compte les liens de perception lors de la phase de résolution. Les conséquences sont directement perceptibles : les robots ont des trajectoires complémentaires, avec un déploiement « en étoile » où chacun apporte au mieux à l'équipe sans gêner les autres par ses actions. Bien évidemment, c'est ce type de comportement qui est souhaitable et cela montre l'importance de modèles précis, riches, et de leur exploitation correcte.

Malgré tout, cela a un revers : les schémas présentés n'ont été calculés que suite à temps de calcul bien plus long – une dizaine de minutes pour des plans très partiels. Bien entendu, un solveur montrant de meilleures performances permettrait de réduire ce temps de calcul mais le problème reste là : l'exploitation de modèles riches est coûteuse, et présente un vrai challenge au niveau de sa complexité.

Par ailleurs, il faut se rappeler que les performances des formulations TOP sont très correctes lorsque l'environnement est structuré comme cela a été montré dans les sections précédentes. On peut alors imaginer utiliser un module de planification décidant à la volée et selon le contexte quelle formulation serait la plus adaptée : TOP pour calculer rapidement avec de nombreux échantillons et dans des environnements structurés, ou plutôt SP pour des plans plus fins et une meilleure coordination en environnement ouvert mais avec une complexité plus grande.

Remarque. Il est possible que les positions de départ pallient partiellement ce problème de redondance des trajectoires, en particulier lorsque les robots sont éloignés les uns des autres au début de la mission. Dans notre contexte, cela reste néanmoins anecdotique car il n'est pas fait de partition de l'environnement au préalable.



(a) $N = 3$ (b) $N = 4$

FIGURE 43 – Planification en formulation SP dans un environnement dépourvu d'éléments structurants (obstacles). On remarque une répartition adéquate de chemins des robots, segmentant l'espace « en étoile ». L'échelle d'utilité est la même pour tous, allant de 0 (noir) à 120 (blanc). ($b = 3$,m et n varient inversement à N)

7.4 INFLUENCE DE L'ÉCHANTILLONNAGE

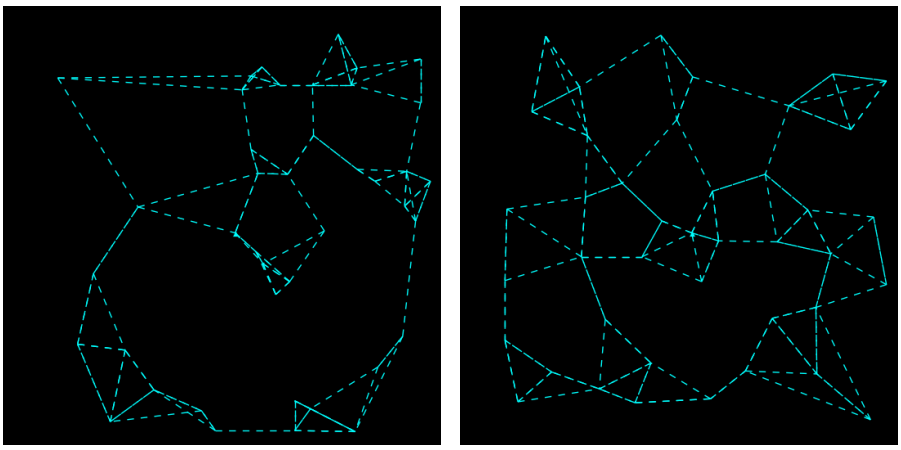
Dans la section 6.3, nous avons mis en avant la volonté de créer des instances par des échantillonnages stochastiques, irréguliers. Il est néanmoins possible de « guider » un peu l'échantillonnage pour obtenir de meilleurs résultats. En particulier, il est important d'essayer de « disperser » spatialement les positions échantillonnées. En effet, lors de la phase d'exécution, ces positions ne sont plus ponctuelles mais occupent plutôt un certain volume – *a minima* celui du robot. En fait, d'une manière générale, il est très difficile d'atteindre très précisément une position, car cela soulève de nombreux problèmes sur le plan du contrôle et de la localisation. En outre, la fonction d'utilité est généralement une fonction continue de l'espace*, et déplacer légèrement les positions échantillonnées n'impacte pas fondamentalement l'instance résultante.

L'utilité u étant une fonction globalement continue de l'espace (continue par morceaux), deux positions proches ont généralement une utilité proche.

Aussi n'est-il pas souhaitable que la planification soit trop précise. En conséquence, on ne cherche pas à choisir entre deux positions échantillonnées proches mais plutôt entre deux alternatives bien distinctes. Cette espacement des échantillons génère une espacement des solutions possibles en découlant et une certaine imprécision souhaitée : il en découle un gain de robustesse comme déjà évoqué dans l'état de l'art.

Les distances minimales entre échantillons sont arbitraires pour Q et proportionnelles à la portée des capteurs pour P^r.

Nous avons donc pris le parti de forcer un peu cette répartition spatiale de la manière suivante : les positions échantillonnées sont choisies de manière stochastique comme décrit à la section 6.3, mais chaque nouvel échantillon doit respecter une certaine distance minimale* avec les autres échantillons du même type. L'effet est illustré à la figure 44.



(a) Distance minimale = 0 (b) Distance minimale > 0

FIGURE 44 – Échantillonnage et stochasticité - (a) présente un échantillonnage stochastique de positions accessibles sans contrainte additionnelle : on peut distinguer plusieurs agglomérats d'échantillons non désirés. (b) présente un échantillonnage stochastique « guidé » par une distance minimale entre les échantillons : le résultat permet une meilleure répartition spatiale des échantillons. ($n = 40$)

Contraindre la stochasticité de l'échantillonnage ne supprime pas pour autant la non-prédictibilité des schémas de patrouille – voir section 7.5.3.

On y perçoit deux échantillonnages à la dispersion spatiale différente : la figure 44a présente un échantillonnage stochastique « pur », au sens il s'agit d'un simple tirage aléatoire selon une distribution de probabilité donnée, sans contrainte additionnelle. On y perçoit différent agglomérats d'échantillons, ce qui n'est pas souhaitable comme expliqué précédemment. La figure 44a présente elle un échantillonnage stochastique* où chaque nouvel échantillon est contraint de respecter une distance minimale aux autres échantillons existants : cela empêche de fait la formation d'agglomérats et a pour effet de bord bénéfique de mieux couvrir la zone à échantillonner. Le résultat final présente toujours des irrégularités comme désiré, ce que nous n'aurions pas obtenu avec un échantillonnage basé sur une grille par exemple.

Remarque. La surface des zones à échantillonner étant naturellement finie, il peut s'avérer impossible de trouver le nombre d'échantillons souhaité lorsque les contraintes de distances minimales entre les échantillons sont trop importantes.

7.5 CYCLES ET PERFORMANCES SUR LE LONG TERME

Les résultats présentés dans cette section ne se concentrent plus sur la résolution d'une instance, et reviennent plutôt au problème d'origine, à savoir la patrouille de zones sécurisées en contexte antagoniste. Dans cette optique, nous présentons d'abord les métriques utilisées ici (section 7.5.1), puis étudions successivement les performances des plans élaborés sur le plan de l'*oisiveté* (section 7.5.2) et de la non-prédictibilité (section 7.5.3).

Remarque. Il n'est bien évidemment pas possible pour des raisons pratiques d'étudier en détails tous les jeux de paramètres possibles et tous les configurations – par exemple les positions de départ. Les résultats ci-après correspondent à des situations que nous avons sélectionnées pour leur pertinence, en gardant à l'esprit que notre objectif est d'étudier les performances de nos algorithmes selon les deux axes « optimalité » et « non-prédictibilité ». En s'appuyant sur notre étude qualitative menée aux sections précédentes, nous avons fortement réduit le nombre de paramètres et de configurations étudiés expérimentalement. En outre, l'étude se concentre sur les formulations TOP car les formulations SP, bien que plus précises, n'offrent pas des performances assez fiables à ce stade pour une utilisation pratique.

7.5.1 *Métriques*

Afin d'étudier correctement les performances de nos algorithmes sur le long terme, nous introduisons deux types de métriques : l'oisiveté nous permet d'évaluer l'optimalité des trajectoires de patrouille tandis que la distance de Fréchet nous permet d'en étudier les (dis)similarités.

7.5.1.1 *Mesure de l'oisiveté*

Le concept d'*oisiveté* (*idleness*) dans les problèmes de patrouille a déjà été présenté dans la première partie – section 2.4. Il correspond peu ou prou au temps écoulé entre deux visites consécutives du même endroit – position ou région – de la zone à patrouiller. Ce critère renvoie directement à la notion de régularité au cœur des problèmes de patrouille. Dans notre cas, nous ne mesurons pas directement l'oisiveté mais plutôt l'utilité qui est associée aux positions : en effet, celle-ci découle directement du concept d'oisiveté mais prend également compte de l'importance relative de la position par rapport aux autres, ce que ne peut faire le concept d'oisiveté seul.

Par la suite, lorsque nous parlons d'« optimalité », nous renvoyons donc directement à l'étude de l'évolution de l'utilité. D'après la fonction objectif, on peut désirer que celle-ci soit aussi basse que possible en moyenne, ou bien par exemple vouloir en minimiser le maximum – voir section 6.6.3.

7.5.1.2 *Distance de Fréchet*

"Given two curves in a metric space, the Fréchet distance between them can be defined intuitively as follows. A man is walking a dog on a leash : the man can move on one curve, the dog on the other ; both may vary their speed, but backtracking is not allowed. What is the length of the shortest leash that is sufficient for traversing both curves ?" - Eiter et Mannula [53]

Afin d'évaluer la non-prédictibilité de nos plans de patrouille, qui est un critère essentiel dans le contexte antagoniste où nous nous plaçons, nous proposons d'utiliser la distance de Fréchet. L'étude de la similarité des trajectoires est un domaine récent mais a déjà fait l'objet de nombreux travaux liés à la quantité massive de bases de données de trajectoires ayant été rassemblées ces dernières années, suite à la démocratisation des systèmes de GNSS [132].

Si la distance de Hausdorff est souvent évoquée, elle présente l'inconvénient de mesurer l'éloignement de deux sous-ensembles d'un espace métrique sous-jacent : elle n'intègre donc aucune notion de séquentialité, pourtant essentielle dans l'étude d'une trajectoire. On lui préfère donc la distance de Fréchet*, et nous renvoyons le lecteur désirant approfondir aux travaux de Eiter et Mannula sur le calcul de la distance discrète de Fréchet [53] qui est une approximation de la distance de Fréchet pour les courbes polygonales – ce qui est exactement notre situation.

7.5.2 *Oisiveté*

Les fonctions objectifs (14), (21) et (22) cherchent à maximiser le gain d'utilité, autrement dit à minimiser l'utilité restante.

Les résultats présentés dans cette section évaluent les performances dans le temps de nos plans de patrouille par rapport à l'oisiveté, c'est-à-dire à la fréquence de visite des différentes zones à patrouiller. On cherche à minimiser* cette oisiveté, ce qui correspond également à minimiser l'utilité sur la carte : en fait, comme expliqué précédemment, l'utilité peut être interprétée comme l'oisiveté pondérée par l'importance de la zone. Il nous est difficile d'évaluer la proximité à l'optimal car celui-ci n'est pas connu pour les problèmes considérés, essentiellement à cause de leur complexité. Nous donnons donc directement les résultats en termes d'utilité moyenne et maximale sur la zone considérée.

Le site de Caylus présente deux types de zones d'importance différentes : les routes accessibles aux AGVs sont pleinement importante, tandis que la zone accessibles à l'AAV uniquement n'est que moitié moins importante.

Nous montrons les résultats pour des suites d'instances avec une croissance d'utilité de 0.1 par unité de temps, pondéré par l'importance de la zone en question*. Nous ne précisons pas l'unité de temps mais il est considéré qu'elle permet de parcourir une unité de distance pour les AGVs. Suivant notre analyse qualitative des sections précédentes, nous nous sommes concentrés sur un tout petit nombre de paramètres : le but n'est pas ici d'affiner les jeux de paramètres avec précision car leur pertinence varie d'un environnement à l'autre. Nous cherchons plutôt à évaluer si les plans calculés sont globalement pertinents au cours du temps. Sauf mention contraire, nos exemples utilisent les paramètres $(\mathrm{N}, \mathfrak{n}, \mathfrak{b}) = (3, 20, 3)$. En revanche, nous faisons varier T dans le but de percevoir son influence. On se demande en particulier s'il est préférable de « planifier une fois pour une période K.T » ou « planifier K fois successivement pour une période T ».

Les figures 45 à 48 présentent l'évolution de l'utilité au fur et à mesure du déroulement des plans de patrouille pour des zones de taille différentes – se référer au tableau 4 en se rappelant que les modèles de Caylus et du LAAS

ont beaucoup de cellules inactives du point du vue des zones accessibles ou observables, ce qui n'est pas le cas des environnements « Plaine » et « Manhattan ». On remarque directement sur les figures 45 et 46 que l'utilité moyenne converge vers une valeur assez faible – en dessous de 50 – pour les environnements restreints de Caylus et du LAAS alors que l'utilité moyenne des environnements « Plaine » et « Manhattan » est croissante et atteint voire dépasse la valeur de 150 – figures 47 et 48.

Ce phénomène se comprend très bien : il correspond au rapport entre la taille de l'équipe de robots patrouillant la zone et la taille de la zone à patrouiller. Dans le cas des environnements plus larges, on s'attend également à converger autour d'une certaine valeur : on observe d'ailleurs que la (dé)croissance de l'utilité ralentit au fur et à mesure du temps. Ainsi, pour une même équipe, la valeur moyenne de convergence semble directement corrélée à la taille des environnements en faisant fi d'autres aspects comme leur topologie. Cela peut paraitre contradictoire avec nos conclusions de la section 7.3.4 indiquant l'influence directrice positive de la topologie de l'environnement. Pourtant, nous considérons ici une suite d'instances et non une seule phase de planification. Or à la fin de cette phase, après exécution ou simulation des plans, l'utilité évolue : là où elle pouvait être homogène, elle devient alors « en relief », ces différences de valeurs créant une véritable topologie artificielle de l'environnement venant s'ajouter à la topologie des zones accessibles, qui est elle intrinsèque. La résolution des formulations IP exploite naturellement cette topologie à travers la prise en compte directe de l'utilité ce qui explique pourquoi même dans des environnements lisses, les formulations TOP peuvent finalement se montrer pertinentes.

Les courbes représentées mettent également en valeur les valeurs maximales obtenues au cours du temps : sans surprise, celles-ci sont globalement croissantes car les robots ne peuvent couvrir en une phase de planification toute la zone à patrouiller, et notamment les positions éloignées de leurs positions de départ. De fait, on perçoit une inflexion de ces valeurs maximales pour les environnements de petite taille comme le LAAS. Tout comme pour les valeurs moyennes, nous supposons que ces courbes tendent à converger vers une valeur dépendant de la taille de l'environnement, avec néanmoins une influence plus notable de sa topologie : l'accessibilité n'est pas « moyennée » ici, et un recoin isolé prend alors toute son importance. Les données des maximums d'utilité sont néanmoins pertinentes : lors de la phase de croissance, ils correspondent à l'évolution qu'aurait l'utilité moyenne sans les observations effectuées par les robots lors de leur patrouille, ce qui met en évidence l'efficacité des plans élaborés.

Remarque. La convergence des valeurs maximales est un effet de bord de la convergence des valeurs moyennes : en effet, conformément aux fonctions objectifs, les valeurs maximales n'ont aucune importance particulière si ce n'est un gain potentiel plus important – voir aussi la section 6.6.3 pour des fonctions objectifs alternatives.

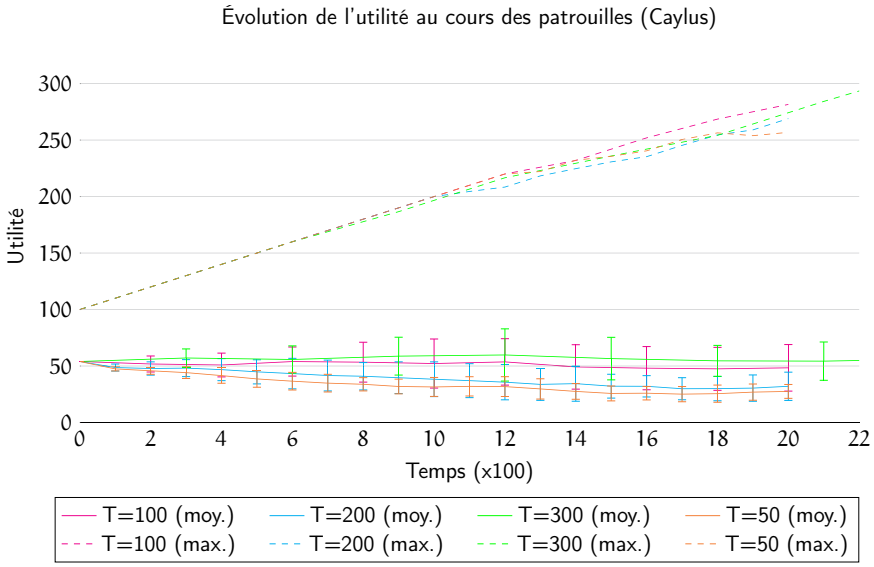


FIGURE 45 – Évolution de l'utilité maximale et moyenne au cours d'une patrouille sur l'environnement « Caylus ». Les résultats sont une moyenne de 20 simulations ; l'écart type entre ces simulations est indiqué à chaque mesure pour l'utilité moyenne. Dans le cas de l'utilité maximale, l'écart type n'est pas indiqué car il est très faible, proche ou égal à zéro. Les couleurs correspondent à différentes valeurs de T, la période de planification, qui désigne aussi le coût maximal des plans des instances.

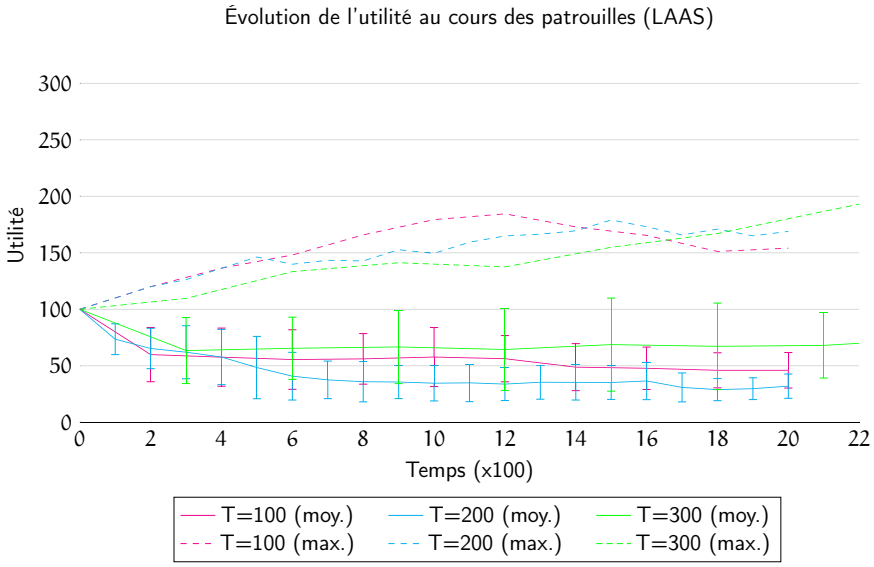


FIGURE 46 – Évolution de l'utilité maximale et moyenne au cours d'une patrouille sur l'environnement « LAAS ». Les résultats sont une moyenne de 20 simulations, présentés comme à la figure 45.

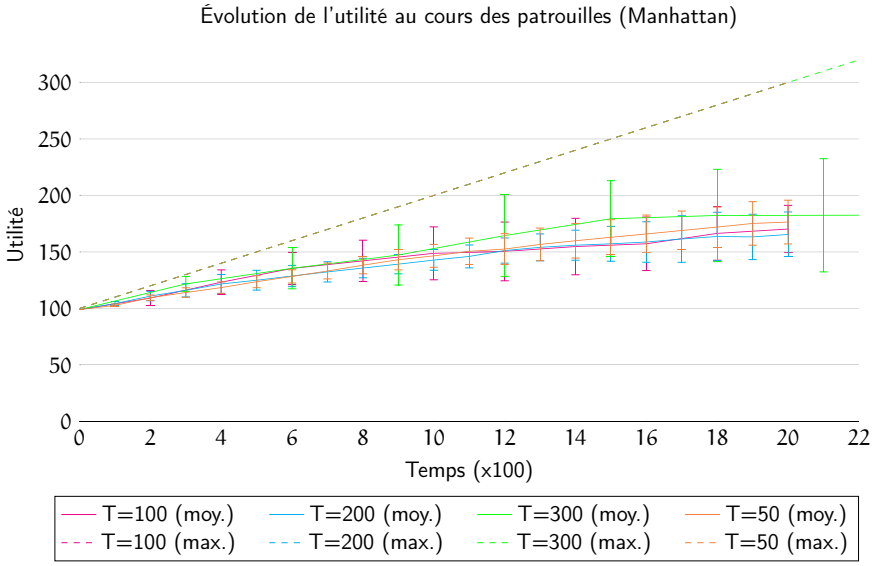


FIGURE 47 – Évolution de l'utilité maximale et moyenne au cours d'une patrouille sur l'environnement « Manhattan ». Les résultats sont une moyenne de 20 simulations, présentés comme à la figure 45.

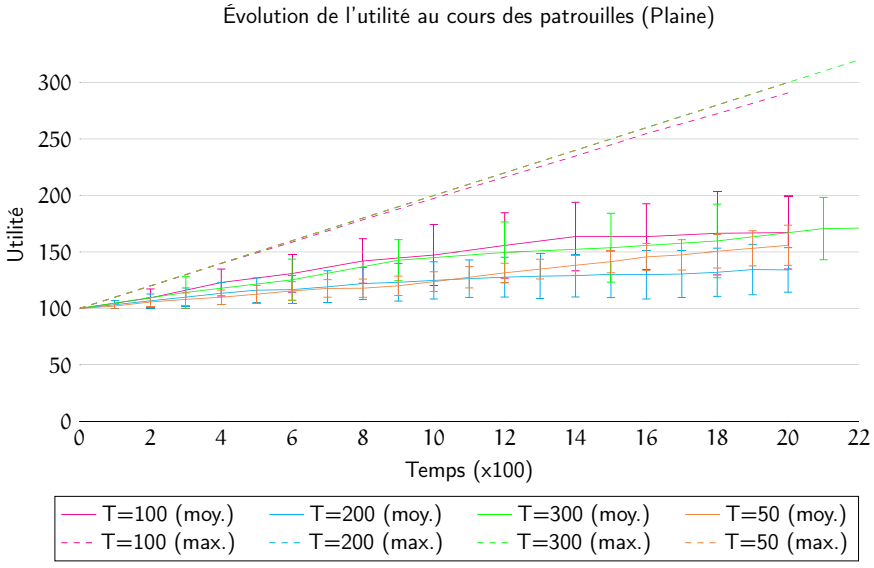


FIGURE 48 – Évolution de l'utilité maximale et moyenne au cours d'une patrouille sur l'environnement « Plaine ». Les résultats sont une moyenne de 20 simulations, présentés comme à la figure 45.

D'après le modèle retenu pour l'évolution de l'utilité, la valeur moyenne de celle-ci grossit en l'absence d'observation d'environ 10 toutes les 100 unité de temps.

De toutes ces considérations, il ressort que les valeurs moyennes observées sont dans l'ensemble très correctes* et globalement stables dans le temps, une fois la convergence atteinte. En d'autres termes, les plans de patrouille élaborés sont pertinents vis-à-vis du critère d'oisiveté. En outre, il faut considérer que l'utilité observée ici est une estimation majorante de l'utilité réelle, comme expliqué à la section 7.1.1. On peut grossièrement estimer que l'approximation dans le calcul de l'utilité en fin d'instance surévalue sa valeur réelle d'environ 10. Par des considérations similaires, on peut estimer dans le cas de Caylus par exemple que l'équipe visite chaque position une fois toutes les 200 unités de temps en moyenne.

Remarque. Les valeurs données dépendent bien évidemment des modèles choisis, notamment le taux de croissance de l'utilité, mais changer raisonnablement les modèles et les valeurs de leurs paramètres ne remet pas en question les comportements généraux observés. Ainsi, même avec une croissance plus forte de l'utilité, on observerait un phénomène de convergence dû au fait que les observations réduisent l'utilité proportionnellement à sa valeur, jusqu'à zéro dans le cas d'une observation parfaite.

Au final, le résultat le plus important – outre l'apparente efficacité des plans calculés – concerne la valeur de T : dans chacun des environnements étudiés, on observe qu'il est plus avantageux d'utiliser une période T petite. En fait, cela se comprend assez facilement en se rappelant que les autres paramètres sont fixes dans notre étude. Il nous faut pour cela considérer la phase d'échantillonnage : la valeur de T indique le coût maximal d'un plan, et définit donc la (non-)proximité des positions accessibles échantillonnées. Or, le nombre de positions échantillonnées est fixe quelle que soit la surface : la densité de l'échantillonnage est donc directement définie par T, en y étant inversement proportionnelle. Un échantillonnage plus dense permet de faire des choix de plans plus subtils et donc d'améliorer la qualité globale des plans, ce qui est observé expérimentalement.

Cela a néanmoins une contrepartie : en parallèle de ce gain de performances, on observe une réduction de l'écart type associé, ce qui signifie que les solutions ont tendance à avoir la même qualité exactement. Bien que cet aspect soit discuté plus en profondeur à la section suivante, on comprend que ce lissage de la qualité induit également un lissage des solutions, c'est-à-dire une perte de diversité dû au caractère stochastique des solutions. On est ici au cœur de notre recherche de compromis entre l'optimalité et la non-prédictibilité. Le fait que T puisse être utilisé comme curseur du compromis optimalité / non-prédictibilité est particulièrement intéressant car cela évite de modifier les autres paramètres de l'échantillonnage – notamment n – et permet donc d'exploiter pleinement les capacités des solveurs.

En complément, et pour mettre en perspective les gains relatifs à la valeur de T, nous proposons d'analyser les résultats de la figure 49 : ils montrent l'impact que l'en peut avoir en faisant varier n au lieu de T sur la

performance des trajectoires vis-à-vis de l'oisiveté. Les gains peuvent être significatifs, comparables à ceux obtenus en faisant varier T. En effet, n n'affecte pas seulement la densité spatiale d'échantillonnage : avant tout, il affecte la complexité du problème IP à résoudre. Le gain en performance est donc directement issus de cette baisse de complexité, permettant au solveur de calculer des solutions optimales en réponse aux instances échantillonnées.

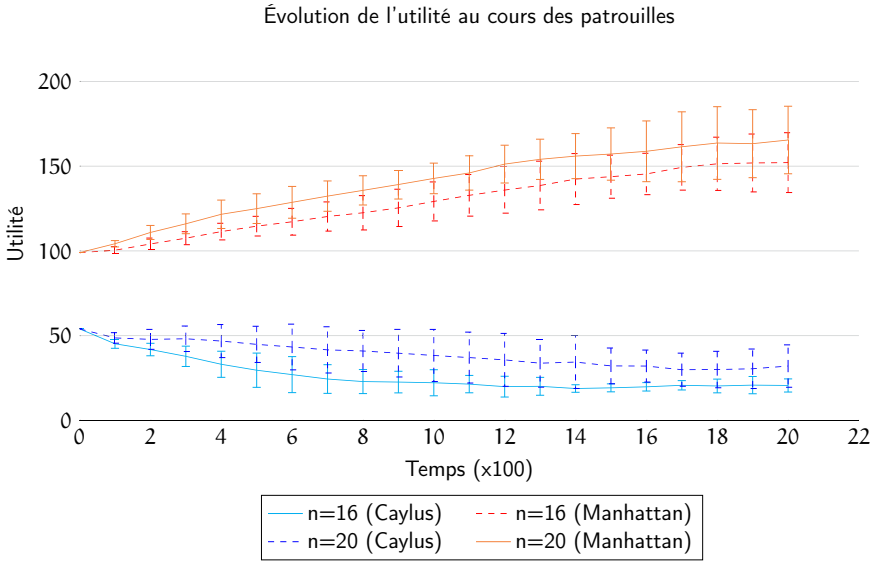


FIGURE 49 – Illustration de l'impact de la complexité – à travers n le nombre d'échantillons – sur les performances des trajectoires de patrouille élaborées – valeurs d'utilité moyenne et écart-types pour $T = 100$.

Remarque. Nos modèles définissent une utilité initiale de 100 environ, avec une évolution de l'ordre de $0.1T \in [5, 30]$ suivant les valeurs de T, soit une progression initiale de 5 à 30%. En pratique, après stabilisation, on observe une utilité moyenne d'environ 50 (au plus) pour les sites du LAAS et de Caylus* et d'environ 150 pour les environnements « Plaine » et « Manhattan ». Pour ces derniers, la croissance d'utilité est donc assez faible en pourcentage, respectant notre hypothèse approximant u par une consante pour la durée d'une instance. En revanche, les environnements de Caylus et du LAAS ne vérifient pas vraiment cette hypothèse ; les algorithmes y montrent néanmoins de bonnes performances, ce qui nous confirme dans l'intérêt de faire cette approximation, même quand elle est grossière. Pour rappel, sans cette approximation, il faut ajouter au problème une composante temporelle complexifiant grandement les formulations, ce qui pose problème lors de la phase de résolution des instances.

Avec $(n, T) = (16, 100)$, on observe une convergence vers des utilités moyennes de 20 environ.

7.5.3 *Non-prédictibilité*

L'objectif de nos plans de patrouille est double : il n'y a pas que leur qualité vis-à-vis de l'oisiveté qui entre en compte, mais aussi leur caractère non déterministe – non prédictible – recherché en réponse au contexte

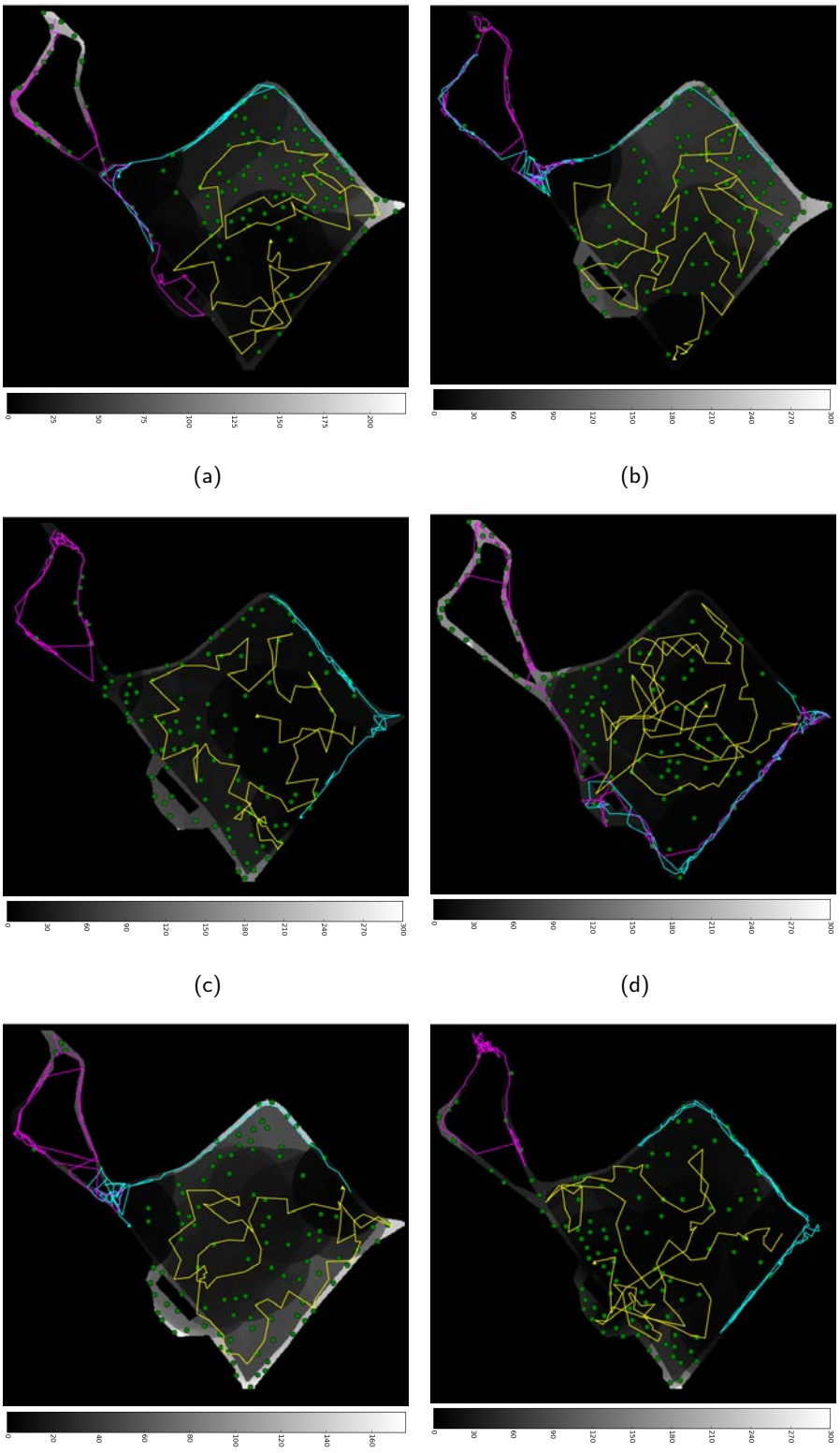


FIGURE 50 – Exemples de plans cumulés, extraits des jeux de données utilisées à la section précédente (T = 100) : on note la grande variabilité apparente, malgré la contrainte des positions de départ.

antagoniste considéré. Nous étudions cet aspect dans cette section. Nous avons déjà vu à la section 7.3.2 qu'il y a une certaine variété entre deux instances calculées à partir des mêmes données initiales. Ce caractère est accentué lors d'une séquence d'instances, comme l'illustre qualitativement la figure 50.

Les êtres humains ayant une mauvaise appréciation de l'aléatoire, nous désirons aller au-delà d'une simple étude qualitative, et fournissons pour cela différentes mesures exploitant la distances de Fréchet. Cette distance permet de mesurer la dissimilarité – ou distance – entre deux trajectoires données. Nous nous en servons ici pour mesurer, à des données d'entrées identiques*, les dissimilarités entre les trajectoires d'un même robot – figure 51 – et entre les trajectoires des différents robots – figure 52.

Les mesures présentées ici sont tirées des tests précédents, en considérant les trajectoires issues de la première instance. Les comparaisons se font donc à données d'entrée et à jeux de paramètres identiques.

La figure 51 montre ainsi qu'à travers les différents environnements, on retrouve une dissimilarité certaine entre les trajectoires dont la valeur moyenne de distance – environ 100 – donne une mesure. On peut également observer un écart type très important, rendant compte de la grande diversité des trajectoires. Ainsi, étant donnés un robot et une situation, notre approche peut donner des trajectoires très différentes.

Bien entendu, cette dissimilarité est à pondérer selon la topologie* du terrain, comme le montre la bien plus faible valeur de dissimilarité pour le robot Minnie dans l'environnement Caylus. Cette valeur s'explique ici par la zone confinée – en haut à gauche – dans laquelle le robot se trouve bloqué. La topologie du terrain peut également accentuer la dissimilarité, comme dans le cas du robot Mana pour Caylus, où seules deux opportunités très distinctes s'offrent au robot – partir vers le haut ou partir vers le bas. Au-delà de l'influence de la topologie, il est notable que la stochasticité de nos algorithmes semble exploiter correctement tout l'étendue des possibilités qui sont offertes. En cela, nos algorithmes sont bel et bien non prédictibles.

La topologie est double : il y a celle intrinsèque à l'accessibilité du robot, et celle découlant de l'utilité. Cette dernière peut être en partie modelée par η lors du processus d'échantillonnage.

Afin de mettre en perspective les mesures de dissimilarité de la figure 51, nous avons calculé les distances de Fréchet moyennes des trajectoires entre robots, présentées à la figure 52. L'intérêt de cette figure est double : dans les environnements de Caylus et du LAAS pour lesquels les conditions de la mission placent les robots dans des positions initiales éloignées, la mesure des distances de Fréchet permet de relativiser les distances entre les trajectoires d'un même robot. En outre, elle montre dans le cas des environnements « Manhattan » et « Plaine » que des robots ayant des capacités similaires présentent les mêmes dissimilarités entre eux que dans leurs propres trajectoires, comme on pouvait s'y attendre sur le plan théorique.

Les comparaisons précédentes s'effectuant à paramètres identiques, nous avons souhaité connaître l'influence de certains d'entre eux. La figure 53 montre les résultats de dissimilarité dans les mêmes conditions que la figure 52 mais pour une valeur de $T = 200$ au lieu de $T = 100$. On note

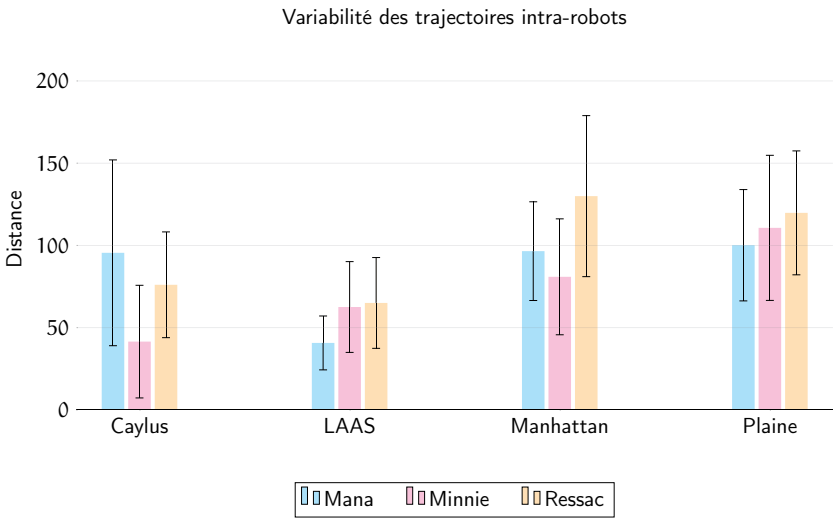


FIGURE 51 – Mesures des dissimilarités entre les trajectoires d'un même robot pour différentes instances d'un même problème de départ, avec données d'entrée et valeurs des paramètres identiques. La distance est calculée à l'aide de la distance de Fréchet : le graphique indique la moyenne et l'écart type des trajectoires pour chaque robot, dans chacun des environnements – les données sont tirées des tests précédemment présentés, avec $T = 100$ et $n = 20$.

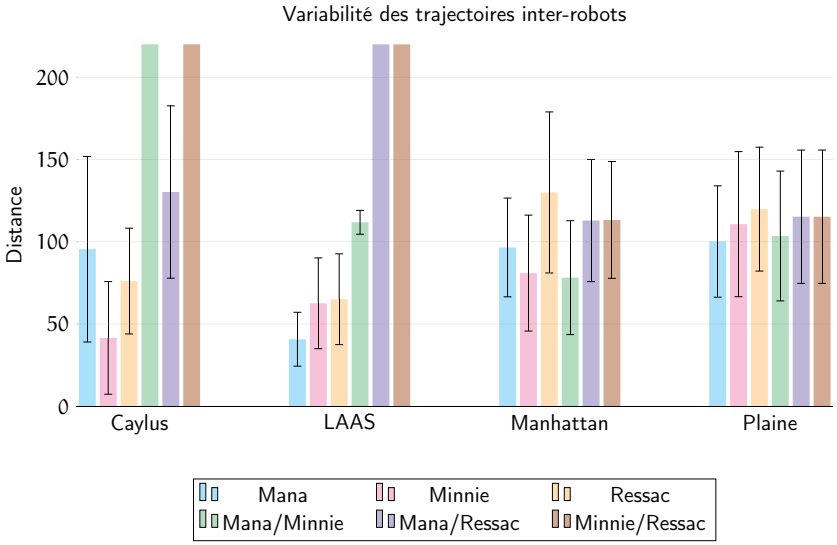


FIGURE 52 – Mesures des dissimilarités entre les trajectoires pour différentes instances d'un même problème de départ, avec données d'entrée et valeurs des paramètres identiques. Le graphique reprend les données de la figure 51 et les complète par des mesures entre les trajectoires des différents robots. Les pics de valeurs dépassant du graphique pointent vers les 400.

une nette progression des dissimilarités dans les valeurs moyennes et dans les écarts types pour des valeurs de T plus élevées, ce qui est conforme à ce que nous attendions. En effet, comme expliqué précédemment, pour n fixé, T sert d'indicateur de densité pour l'échantillonnage. Or un échantillonnage moins dense mène à des plus grandes disparités possibles dans les positions échantillonnées, ce qui résulte en des trajectoires très différentes les unes des autres. Les résultats observés ne font donc que confirmer nos propos précédents : T agit bien comme un curseur entre optimalité et non-prédictibilité.

Remarque. À T fixé et en faisant varier n faiblement, par exemple de 20 à 16 ou à 25, les résultats des mesures de dissimilarités ne changent pas sensiblement. En revanche, cela se ressent fortement sur les performances du solveur au niveau du nombre de solutions valides effectivement renvoyées. Ceci est un argument de plus en faveur de l'exploitation de T plutôt que n comme curseur de l'optimalité et de la non-prédictibilité.

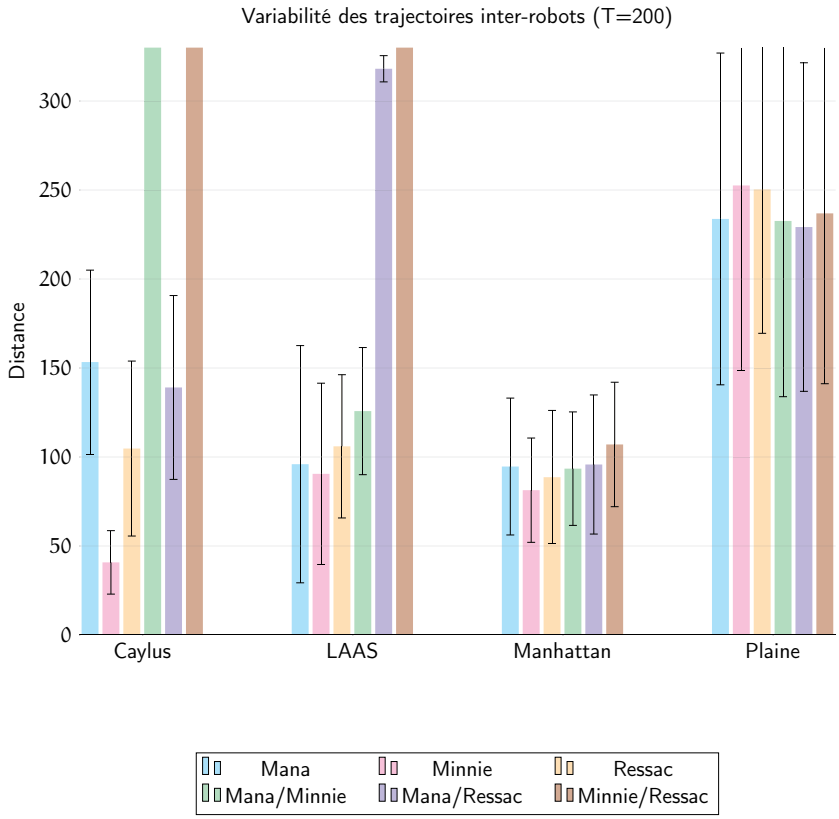


FIGURE 53 – Mesures des dissimilarités entre les trajectoires pour différentes instances d'un même problème de départ. Le graphique est similaire à celui de la figure 51 mais présente des résultats pour $T = 200$ au lieu de $T = 100$ – les données sont également tirées des tests précédents.

7.6 COMMUNICATION

Considérant nos discussions sur les contraintes de communication et la complexité associée (section 6.6.4), nous n'avons pas implémenté les communications dans le cas d'une résolution centralisée. Ainsi, le lecteur est renvoyé à la section 8.4 présentant l'impact des communications dans une approche décentralisée, les résultats présentés pouvant être étendus au cas centralisé.

7.7 BILAN ET PERSPECTIVES

Nous avons testé tout au long de ce chapitre différents aspects de nos algorithmes, les comparant entre eux, évaluant les jeux de paramètres pertinents, leurs marges de manœuvres et les performances finales des plans de patrouilles élaborés. Nous pouvons résumer nos résultats en quelques points. Tout d'abord, nos algorithmes fonctionnent bien : les plans élaborés sont intéressants et couvrent effectivement toute la zone sécurisée. En particulier, nous avons vu que nos algorithmes permettent effectivement un compromis entre l'optimalité des schémas de patrouille et leur non-prédictibilité. Ils répondent donc à nos objectifs premiers et valident par la même les hypothèses et approximations les soutenant.

Tout au long des tests expérimentaux, nous avons avons été confrontés aux limites des performances du solveur IP. En effet, les problèmes TOP et SP abordés étant intrinsèquement complexes, il est facile d'augmenter les valeurs des paramètres pour dépasser les capacités du solveur. Au regard de ce premier prototype aux performances prometteuses, nous désirons implémenter nos algorithmes dans un solveur beaucoup plus performant. Aux travers de nos tests, nos avons néanmoins pu confirmer nos premières impressions sur les différentes formulations : les formulations TOP* sont tout à fait pertinentes et efficaces dans la plupart des situations réelles mais montrent cependant leurs limites en terme de qualité des solutions face aux formulations SP. Cela se remarque notamment dans les environnements ouverts où la coordination entre robots ne s'appuyer sur la topologie environnante. Malgré tout, le coût prohibitif en calcul des formulations SP limite pour l'instant leur utilisation, du moins jusqu'à une nouvelle implémentation plus performante.

Nous avons testé essentiellement testé les formulations TOP-per et non celles TOP-pos, car nous pouvons voir ces dernières comme un cas particulier des premières pour lesquelles la fonction de visibilité ϕ^r est un Dirac.

Il est apparu que la meilleure stratégie pour fixer les paramètres était de s'appuyer sur les limites du solveur. Nous pouvons ensuite fixer la période d'instance T, qui n'a qu'une influence limitée sur les capacités du solveur IP. T sert alors de curseur entre l'optimalité et la non-prédictibilité des plans élaborés, et d'une manière générale permet d'ajuster les performances des algorithmes. Tous ces paramètres sont en revanche à fixer de manière empirique : la plupart par rapport au solveur, et T par rapport à la mission. En particulier, la topologie des zones à patrouiller influe beaucoup sur les performances constatées.

Nous avons effectué tous nos tests à l'aide de simulation *ad hoc*. L'étape suivante consiste à tester avec un simulateur réaliste comme Morse. Cela devrait se faire sans trop de complications car nous exploitations déjà des modèles réalistes. Nous avons par ailleurs essayé de fournir une première analyse statistique de nos résultats. Comme expliqué au début de ce chapitre, il n'est pas possible de tout tester systématiquement, mais nous avons malgré tout essayé de fournir des résultats aussi larges et variés que possible, en cherchant également une certaine redondance venant renforcer les résultats majeurs comme le rôle de T. Nous regrettons cependant de ne pas pouvoir donner de bornes théoriques sur la sous-optimalité des solutions. Peut-être qu'une analyse théorique profonde de l'étape d'échantillonnage permettrait de venir palier ce manque.

De plus, nous avons négligé jusqu'ici l'impact des communications. Celles-ci apportent en effet un niveau de complexité supplémentaire que nous n'avons pas intégré à ce stade. Nous en testons différents aspects dans le chapitre suivant, traitant de nos algorithmes dans une version décentralisée.

Enfin, il reste de nombreuses questions plus larges, traitant de la supervision des processus de planification présentés ici. Bien que cela sorte du strict cadre de la planification, c'est un aspect essentiel dans le processus d'intégration des algorithmes, et nous tenons ici à mentionner quelques uns des problèmes soulevés en donnant parfois des pistes de solution. Le premier constat est que le solveur peut échouer : bien sûr, le choix des valeurs des paramètres influençant la complexité va grandement fixer ce taux d'échecs jusqu'à le rendre presque nul, mais il peut néanmoins échouer. Cela ne dépend d'ailleurs pas que du solveur : il est théoriquement possible que l'échantillonnage soit tel qu'on ne puisse en tirer aucune trajectoire faisable, car celui-ci ne gère que des positions et non leurs liens entre elles. Dans ces situations, comment doivent réagir les robots ? Il leur faut bien sûr replanifier, mais faire cela en restant statique nous semble peu adapté au contexte antagoniste.

Dans le même ordre d'idées, les solutions renvoyées sont parfois comme incomplètes – dans nos exemples c'est surtout le cas avec les formulations SP. Il nous faut alors d'une part détecter ces situations, et ensuite décider de comment réagir. Détecter ces solutions bancales est relativement simple car au moins un des robots y sous-exploite le temps T alloué, c'est-à-dire que le plan du robot ne demande qu'une fraction de T pour être exécuté. Le solveur donne également des indications additionnelles sur la qualité des solutions. Une réaction possible est de jouer les plans « tronqués » pour s'aligner sur le robot avec le plan le plus cours, mais cette solution n'est que peu satisfaisante et risque de ne pas respecter certaines contraintes respectées par les solutions entières – même en étant bancales.

En l'absence de solutions pertinentes à ces problèmes, leur fréquence peut facilement être limitée en choisissant des paramètres adaptés aux solveurs, et en lançant plusieurs instances du planificateur de trajectoires : la redondance permet alors de couvrir la plupart des cas d'échecs, si ceux-ci ne sont

pas trop fréquents pour apparaître simultanément ; mais cela reste coûteux en ressource et peu satisfaisant.

La considération des temps de calcul nécessaires à la planification est également un aspect important pour une intégration dans des scénarios réalistes. En particulier, nous avons indiqué avoir limité nos temps de calcul à moins de 300 secondes – soit cinq minutes – pour effectuer nos tests. En pratique, cela reste encore trop long : au regard du type de mission et de la longueur des plans, il n'est pas envisageable de simplement « attendre » cinq minutes que le solveur ait tout calculé. Bien entendu, l'intégration plus poussée de nos algorithmes sur un solveur performant permettra de réduire drastiquement ces temps de calcul afin de revenir sur des bornes plus acceptables de quelques dizaines de secondes au maximum*. Au cas où cela ne suffirait pas, ou bien au cas où l'on préférerait augmenter les valeurs des paramètres plutôt que réduire les temps de calculs, il nous semble envisageable d'effectuer une planification en deux temps : pendant l'exécution d'un cycle, il est possible de calculer le suivant. Bien entendu, cela suppose de planifier sur des données d'utilité simulées plutôt que sur celles issues de l'exécution du plan, mais cela semble raisonnable. En fait, la pertinence de cette approche est surtout à évaluer aux regards des aléas survenant dans la mission, et en particulier des écarts entre les plans élaborés et leur exécution. Si ces aléas ne sont pas nombreux ni importants, alors cette approche en deux temps est intéressante. Dans le cas contraire, les plans élaborés seront trop éloignés de la réalité pour être pertinents.

Les bornes d'acceptabilité des temps de calcul de la planification sont avant tout à considérer relativement à T*, la durée d'exécution d'un cycle de patrouille.*

Tous ces aspects de supervision n'ont été qu'effleurés dans nos travaux ; nous y revenons partiellement au chapitre suivant concernant l'élaboration de schémas de patrouille de manière décentralisées, ainsi que la discussion finale.

8 PATROUILLES DÉCENTRALISÉES

Anarchie : *du grec ἀναρχία / anarkhia, composé de an, préfixe privatif : absence de, et de arkhê, hiérarchie, commandement.*

L'anarchie, c'est l'ordre sans le pouvoir.

— Proudhon

Dans ce chapitre, nous présentons nos résultats expérimentaux obtenus en implémentant une version décentralisée des algorithmes de patrouille présentés au chapitre 6. Ces résultats sont à mettre en perspective avec ceux présentés au chapitre 7, correspondant à la version centralisée. En particulier, nous reprenons les mêmes éléments d'analyse : modèles, données, légendes, environnements de tests, métriques, *etc.* Ce chapitre traite d'abord des conséquences de la décentralisation sur la résolution des problèmes d'optimisation en nombres entiers – section 8.1. Il met ensuite en évidence le besoin de coordination entre les robots – section 8.2 – avant de comparer plus quantitativement les performances obtenus dans nos versions décentralisées par rapport aux versions centralisées – section 8.3. Nous étudions enfin l'impact de la prise en compte des communications sur notre approche – section 8.4. Une mise en perspective conclut ce chapitre.

8.1 DÉCENTRALISATION ET RÉSOLUTION IP

Dans la partie I, nous avons cité plusieurs approches décentralisées abordant différents problèmes de détection et de suivi de cibles. Comme rappelé dans le bilan de cette partie, nous percevons les approches sous-optimales mais efficaces comme une réponse appropriée à la complexité intrinsèque des nombreux problèmes de planification, dont les formulations TOP et SP proposées pour résoudre les problèmes de patrouille. Par « efficace », nous entendons « qui produit des solutions de bonnes qualités dans les contraintes imparties ». Pour des applications réelles, ces problèmes sont essentiellement la limitation du temps de calcul autorisé et la robustesse aux aléas et imprécisions ; cela inclut la gestion d'un environnement dynamique.

Nous pensons que les algorithmes décentralisés répondent pour la plupart à ces attentes. En particulier, ils sont naturellement résilients face aux aléas et aux échecs, car ce sont des systèmes composés dont la perte d'un élément ne change pas le comportement global. En effet, les algorithmes décentralisés sont distribués entre les robots, c'est-à-dire que les calculs sont, dans le cas idéal, répartis équitablement entre tous les robots, sans que l'un d'entre eux soit plus important que les autres. Ces algorithmes présentent deux avantages importants : d'une part ils ne sont pas censés avoir

de centres névralgiques, c'est-à-dire que si un des robots subit un aléa grave le mettant « hors service » – une panne par exemple – alors cela n'aura qu'un impact négligeable sur le comportement des autres agents. D'autre part, la charge de calcul étant répartie, on peut rajouter des robots sans craindre une explosion de la complexité. Beaucoup d'entre eux ont alors une complexité constante ou linéaire en le nombre de robots, ce qui signifie qu'ils satisfont beaucoup mieux les contraintes de calcul, voire celles de communication car ces algorithmes nécessitent rarement de partager toutes les données. Cette dernière propriété est au moins aussi importante que la première, mais c'est la plus difficile à obtenir en pratique car elle suppose que les calculs peuvent être séparés en différentes parties indépendantes les unes des autres.

Nous proposons dans ce chapitre d'adapter notre méthode pour la transformer en un processus décentralisé. Notre approche ayant été conçue à l'origine comme un processus centralisé, cette adaptation n'est pas directe ; en particulier, elle ne présente pas comme nous le souhaiterions toutes les propriétés d'un processus de décision décentralisé idéal. En adaptant les différentes étapes de notre processus de décision, nous proposons deux méthodes décentralisées que nous étudions par la suite : une première méthode que nous qualifions de « décentralisée parallèle » et une autre méthode que nous qualifions de « décentralisée séquentielle ».

Dans notre méthode décentralisée parallèle, nous proposons simplement que chaque robot effectue lui-même un cycle de planification – instanciation et résolution IP – en considérant une équipe de robots restreinte à lui seul. Ainsi, les robots partagent leurs informations – et notamment les utilités – mais planifient en solitaire. Cette méthode présente l'avantage d'être entièrement distribuée entre les robots : le temps global de calcul est donc constant quel que soit le nombre d'agents dans l'équipe. C'est une propriété extrêmement intéressante dans notre contexte où le temps de calcul est restreint. En outre, chaque robot étant responsable de ses propres plans, le comportement global de l'équipe est robuste aux aléas, et dans une certaine mesure aux environnements dynamiques. Le revers de cette approche est qu'il n'y a aucune forme réelle de coordination entre les robots. Ainsi, hormis lorsque la topologie du terrain et les positions des robots sont favorables, on peut craindre une chute drastique des performances. Cette chute est néanmoins difficilement quantifiable *a priori* à cause de des aspects stochastiques et topologiques qui pourraient compenser ce manque de coordination.

Dans notre méthode décentralisée séquentielle, chaque robot planifie pour lui-même mais en séquence, c'est-à-dire qu'un ordre arbitraire est donné aux robots, et que ceux-ci planifient lorsque que les robots placés avant ont fini de calculer leurs plans respectifs. L'avantage par rapport à la méthode précédente est la présence d'une coordination plus explicite entre eux, chaque robot prenant en compte les plans des robots précédents. Cette coordination a néanmoins un coût : le processus de décision est maintenant linéaire

avec le nombre de robots. Néanmoins, cette linéarité ne s'exprime pas de la même manière que pour notre méthode centralisée. En effet, cette dernière correspond à la résolution d'une instance IP de complexité $O(N.K)$ – avec K un paramètre abstrait rendant compte de la complexité des autres paramètres et N le nombre de robots. Notre méthode décentralisée séquentielle requiert, elle, la résolution de N instances IP de complexité $O(K)$. Or, il s'avère que la résolution d'un problème IP n'est pas linéaire avec sa complexité : il est plus simple et plus rapide de résoudre N instances de complexité $O(K)$ plutôt qu'une seule de complexité $O(N.K)$ – voir annexe B. Cette particularité des problèmes IP rend donc notre méthode séquentielle avantageuse sur le plan du temps de calcul et de l'étendue des jeux de paramètres acceptables – il est notamment possible d'avoir plus d'échantillons.

Ce gain en complexité et en temps de calcul, commun aux deux méthodes décentralisées, nous permet d'envisager l'exploitation des formulations SP, trop complexes en centralisé mais qui semblent abordables ici, où l'on ne résout que des instances à un seul robot – en parallèle ou en séquence. En outre, cette baisse de la complexité globale nous permet d'envisager l'ajout des contraintes de communications, ce que nous n'avions pas fait en centralisé.

Dans la suite de ce chapitre, nous tentons dans en premier temps de lever le doute sur la pertinence de notre approche décentralisée parallèle. Nous comparons ensuite les performances à long terme avec celles obtenues dans la version centralisée. Nous étudions ensuite l'introduction de contraintes communications et leur impact sur le comportement global de l'équipe. Nous concluons ce chapitre avec une discussion des perspectives.

8.2 LE BESOIN DE COORDINATION

Dans cette section, nous cherchons à comparer les performances de nos approches décentralisées, parallèle et séquentielle. En particulier, nous souhaitons vérifier ou infirmer notre crainte d'une chute des performances liées à l'absence totale de coordination dans la version parallèle. Pour cela, nous fournissons une analyse des performances en matière d'oisiveté pour les approches parallèle et séquentielle respectivement, de manière analogue à la section 7.5.2*. Nos résultats expérimentaux sont résumés par les figures 54 à 57 comparant les performances des deux approches sur chacun de nos quatre environnements de tests.

La limite de temps de calcul est néanmoins fixée à 60s par robot en séquentiel, et 60s en tout en parallèle, contre 300s en centralisé.

Remarque. Nous étudions ici le besoin (ou non) de coordination entre les robots, c'est pourquoi nous avons choisi de se concentrer sur la mesure de l'oisiveté, laissant la non-prédictibilité de côté pour l'instant. De plus, nos graphiques présentent uniquement la moyenne des utilités moyennes obtenues car nous avons vu précédemment que les utilités maximales n'étaient pas significatives dans les tests de performances, conformément à la fonction objectif retenue. Les résultats concernant les utilités maximales sont d'ailleurs sensiblement identiques à ceux obtenus en centralisé.

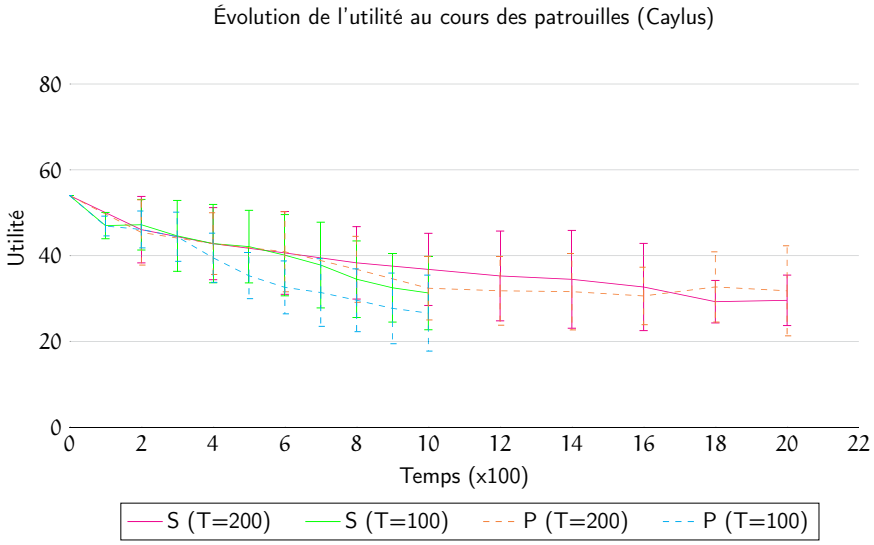


FIGURE 54 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « Caylus ». Les résultats sont une moyenne de 20 simulations ; l'écart type entre ces simulations est indiqué à chaque mesure. Ce graphique compare l'approche décentralisée parallèle (P) et l'approche décentralisée séquentielle (S) à paramètres égaux ($N = 3$, $n = 40$, T varie).

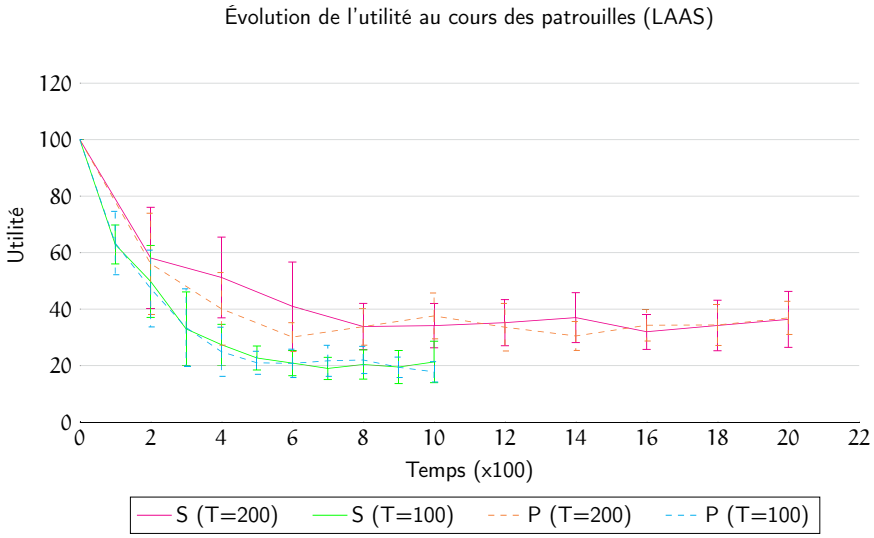


FIGURE 55 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « LAAS ». Les résultats sont présentés de la même manière qu'à la figure 54.

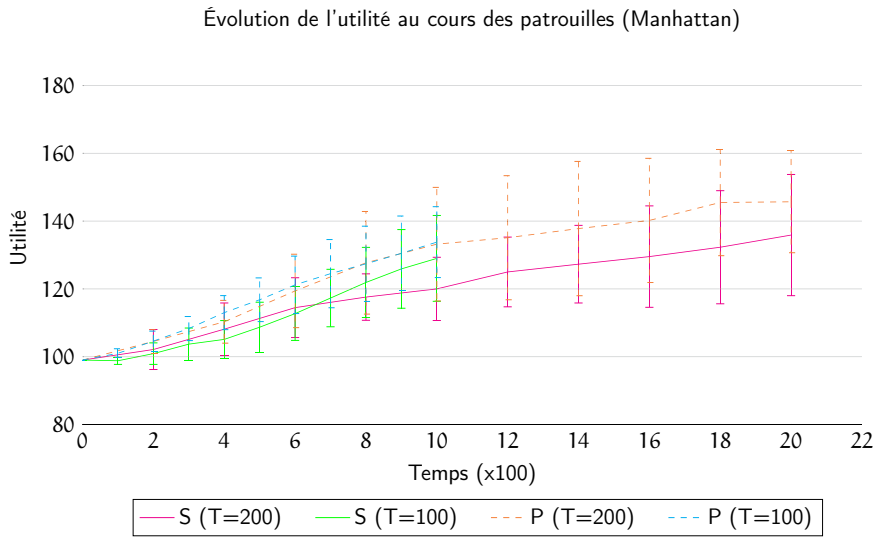


FIGURE 56 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « Manhattan ». Les résultats sont présentés de la même manière qu'à la figure 54.

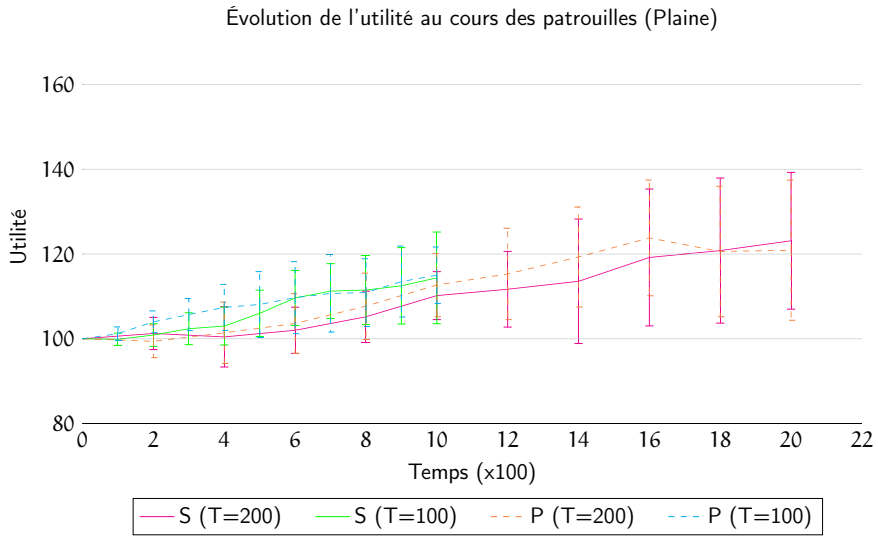


FIGURE 57 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « Plaine ». Les résultats sont présentés de la même manière qu'à la figure 54.

Il est tout de suite remarquable que les deux approches décentralisées – séquentielle et parallèle – présentent des performances tout à fait similaires sur les environnements « Caylus », « LAAS » et « Plaine ». Seul l'environnement « Manhattan » départage les performances des deux approches, au profit de la version séquentielle.

Ce résultat est de premier abord assez contre intuitif, car il semble suggérer que la coordination entre les robots n'apporte pas de gain significatif. Il faut alors se rappeler les configurations de départ de ces environnements présentés dans le chapitre précédent (figures 38, 39, 41 et 42), et qu'il est difficile d'évaluer *a priori* la chute des performances due à l'absence de coordination à cause des aspects stochastiques et topologiques.

Dans le cas des environnements « Caylus » et « LAAS », les configurations retenues placent les robots relativement éloignés les uns des autres, de sorte qu'il n'est pas nécessaire de les coordonner entre eux au départ. La bonne nouvelle vient du fait qu'il semble que cette situation se prolonge dans le temps, les robots restant dispersés dans l'environnement. En d'autres termes, la situation initiale, puis le déroulement des plans et la topologie de l'environnement* génèrent une partition naturelle de l'environnement dans laquelle chacun des robots peut évoluer indépendamment des autres.

Pour rappel, nous rassemblons sous la notion de topologie celle intrinsèque à l'environnement – les obstacles – et celle induite par l'utilité.

Pour les environnements « Manhattan » et « Plaine », cette partition naturelle n'existe pas au départ car les robots se trouvent initialement regroupés. On voit ainsi l'influence directe sur les performances dans l'environnement Manhattan, où la version parallèle montre de moins bonnes performances. Le déséquilibre reste néanmoins limité pour la durée considérée – de 10 à 20% de surplus d'utilité. En revanche, les performances semblent à nouveau sensiblement identiques pour l'environnement « Plaine » : nous attribuons cela au fait que d'une manière générale les formulations de type TOP montrent de piètres performances dans ce type d'environnement non structuré – voir la section 7.3.4 ; le gain de performances lié à la coordination y reste donc limité.

En résumé, les performances de l'approche décentralisée parallèle souffrent bel et bien du manque de coordination entre les robots dans cette approche. Il existe néanmoins de nombreuses situations pour lesquelles ce manque est réduit voire entièrement palié. Dans ces cas-là, les deux approches séquentielle et parallèle montrent des performances tout à fait similaires. Considérant les autres atouts de l'approche parallèle et notamment en temps de calcul, nous pensons que cette approche ne doit pas être mise de côté. Malgré son caractère situationnel, qui par ailleurs reste fréquent, elle est très avantageuse.

Remarque. Cette section illustre à nouveau le besoin d'utiliser des environnements de tests variés et des conditions de départ différentes, permettant de mettre en évidence différentes situations et donc de mieux évaluer les performances de chaque approche.

8.3 ASPECTS LONG TERME

La section précédente a mis en évidence le potentiel de nos deux approches décentralisées. Nous souhaitons ici approfondir cette étude en comparant ces deux approches non pas entre elles, mais plutôt avec les performances obtenues en centralisé – voir section 7.5. Nous étudions ici les performances sur le long terme et nous reprenons les métriques précédemment introduites à la section 7.5.1 : la mesure de l'oisiveté à travers celle des utilités et la mesure de la non-prédictibilité à travers celle des distances de Fréchet entre les trajectoires.

Nous abordons ici plusieurs aspects : d'abord les performances vis-à-vis de l'oisiveté des formulations TOP résolues en décentralisé – section 8.3.1 – puis leurs performances vis-à-vis de la non-prédictibilité – section 8.3.2 – et les performances des formulations SP – section 8.3.3. Ces dernières n'avaient jusqu'ici pas été évaluées à cause de la grande complexité de ce type de formulation causant un taux d'échecs du solveur trop important en centralisé dans les conditions de temps de calcul imposées. Nous clôturons ces aspects longs termes par une mention aux aspects très long termes – section 8.3.4.

Remarque. Sauf mention contraire, nous n'accordons au solveur qu'une minute de temps de calcul pour résoudre les problèmes IP, ce qui donne une temps de calcul cumulé de trois minutes en séquentiel et d'une minute en parallèle. Ce temps de calcul est à comparer avec celui utilisé en centralisé – cinq minutes ; cette considération permet de mettre en perspective les jeux de paramètres utilisés et les performances associées.

8.3.1 *Oisiveté*

Dans cette section, nous comparons les performances des approches centralisées avec celles des approches décentralisées sur le plan de l'oisiveté. Pour cela, nous avons rassemblé sur un même graphique – figures 58 à 61*– les résultats des sections 7.5.2 – en centralisé – et 8.2 – en décentralisé. Afin de ne pas nuire à la lisibilité des figures, nous avons néanmoins enlevé les barres d'erreur ; il suffit de se reporter aux figures précédentes pour les connaitre. Leur prise en compte permet de relativiser les écarts observés – cela les rend moins importants ; les propos tenus par la suite en tiennent compte.

Dans ces figures, nous utilisons les paramètres $(N = 3, n = 20)$ *en centralisé et* $(N = 3, n = 40)$ *en décentralisé ;* T *varie.*

Remarque. Pour rappel, l'oisiveté est un critère classique utilisé pour évaluer la pertinence des plans de patrouille en matière d'« optimalité ». On cherche à la minimiser.

De premier abord, les résultats observés peuvent sembler surprenants : il apparaît en effet que les approches décentralisées sont au moins aussi performantes que l'approche centralisée. Pourtant, nous savons que, par construction, les approches décentralisées ne résolvent pas optimalement

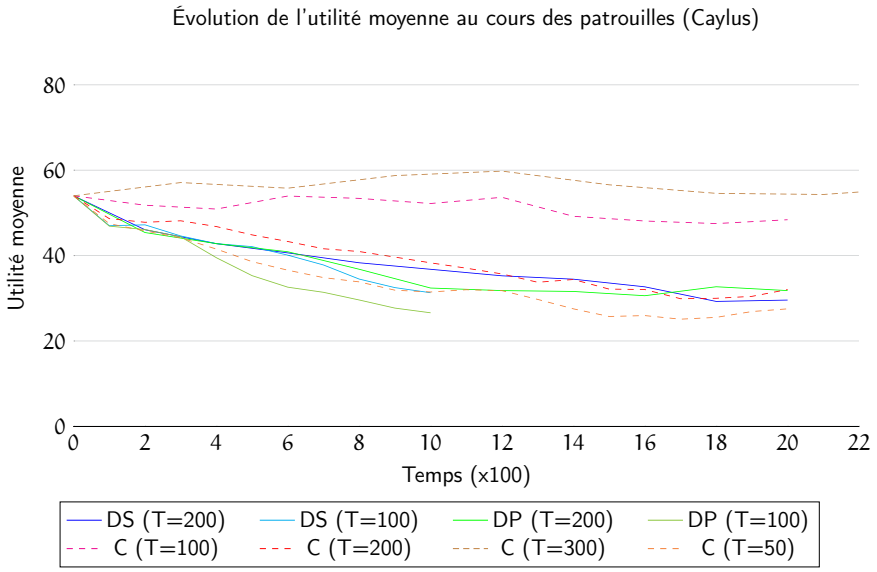


FIGURE 58 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « Caylus ». Ce graphique rassemble les résultats des figures 45 et 54, et permet de comparer les approches décentralisées parallèle (DP) et séquentielle (DS) à l'approche centralisée (C) pour des formulations TOP.

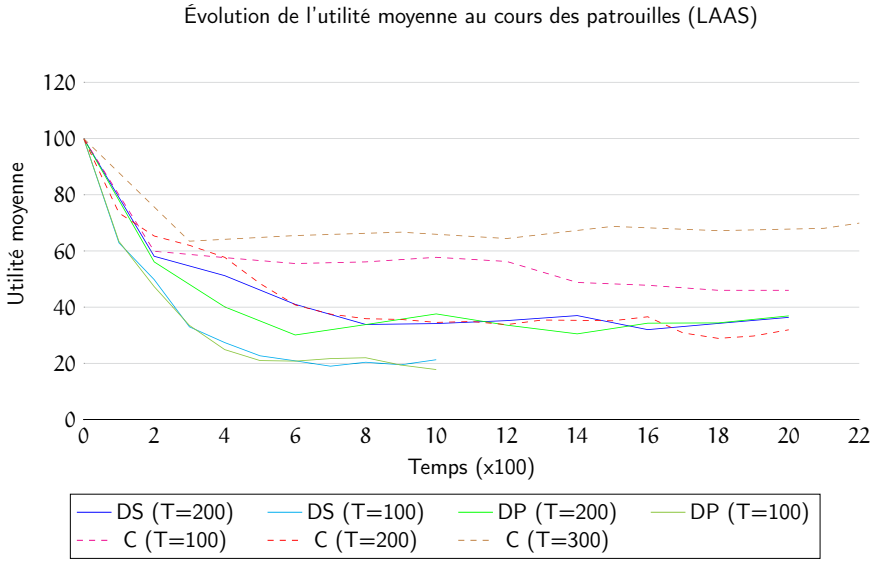


FIGURE 59 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « LAAS ». Ce graphique rassemble les résultats des figures 46 et 55, et permet de comparer les approches décentralisées parallèle (DP) et séquentielle (DS) à l'approche centralisée (C) pour des formulations TOP.

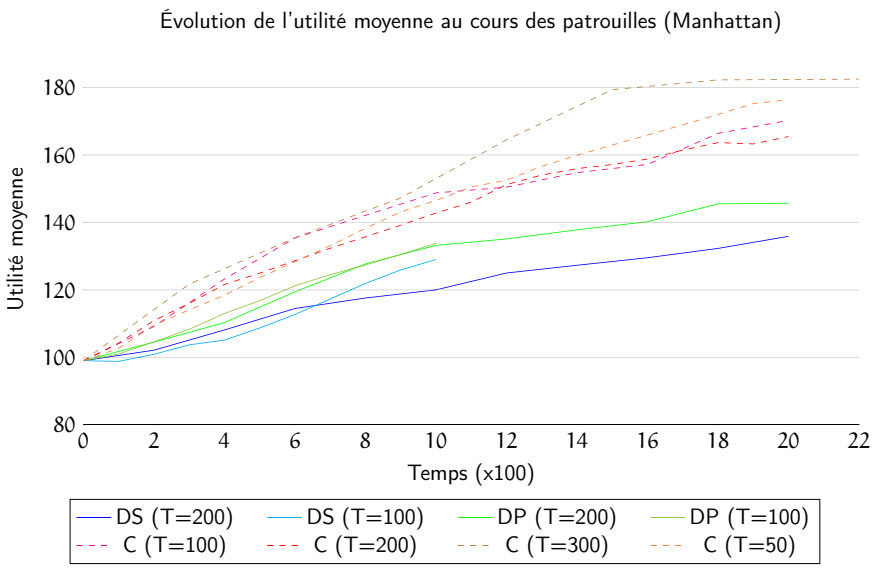


FIGURE 60 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « Manhattan ». Ce graphique rassemble les résultats des figures 47 et 56, et permet de comparer les approches décentralisées parallèle (DP) et séquentielle (DS) à l'approche centralisée (C) pour des formulations TOP.

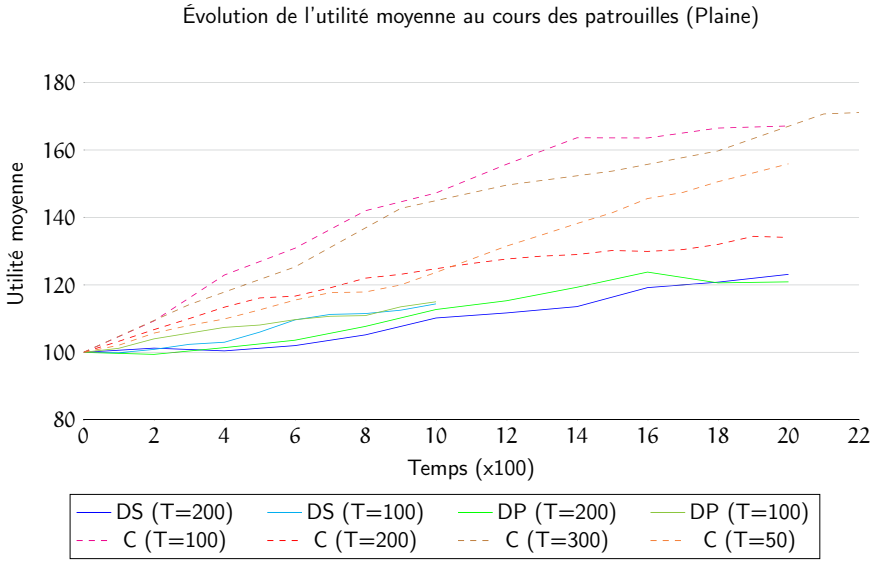


FIGURE 61 – Évolution de l'utilité moyenne au cours d'une patrouille sur l'environnement « Plaine ». Ce graphique rassemble les résultats des figures 48 et 57, et permet de comparer les approches décentralisées parallèle (DP) et séquentielle (DS) à l'approche centralisée (C) pour des formulations TOP.

la totalité de l'instance, mais résolvent à la place plusieurs sous-instances – une par robot – parfois même sans coordination entre les robots. Nous expliquons ces résultats par les bénéfices indirects de nos approches centralisées, et en particulier par le gain de flexibilité au niveau des paramètres auquel viennent s'ajouter les explications de la section précédente sur l'importance des situations initiales sur le besoin ou non de coordination.

En effet, nous avons vu précédemment à la section 7.5.2 qu'une plus grande densité d'échantillons – obtenue alors en réduisant T à $\mathfrak{n}$ constant – permet d'obtenir de meilleures performances globales. Or nos approches décentralisées permettent d'augmenter de manière importante cette densité d'échantillonnage car elles présentent une complexité moindre ; on a constamment $N = 1$ dans les problèmes IP à résoudre – se référer à la section 7.2.1 pour le détail des complexités. Dans nos exemples, la densité des positions accessibles échantillonnées est systématiquement doublée pour les approches décentralisées – $\mathfrak{n} = 40$ au lieu de $\mathfrak{n} = 20^*$.

Nous avons pu doubler la densité de l'échantillonnage en décentralisé tout en réduisant le temps de calcul car les instances IP à résoudre sont, en définitive, plus simples pour le solveur – on a $N = 1$.

Il est vrai qu'en décentralisé les trajectoires des robots ne sont plus calculées conjointement mais élaborées en séquence ou en parallèle, c'est-à-dire qu'au moins une partie des trajectoires sont définies en ignorant ce que font les autres robots. Nous pouvons néanmoins étendre nos propos de la section 8.2 à propos des situations ne nécessitant pas une étroite coordination : dans celles-ci, la densité de l'échantillonnage est bien plus importante que le fait d'élaborer conjointement les trajectoires – c'est notamment le cas pour les environnements « LAAS » et « Caylus ».

De plus, la séquentialité est aussi une réponse au problème soulevé à la section 7.3.4 à propos de la redondance de la perception ignorée pour les formulations TOP. En effet, par exemple pour Manhattan ou pour Caylus, le fait que les robots planifient en séquence permet de prendre en compte la perception des robots ayant déjà planifié une trajectoire. Ceci semble éclairer les résultats présentés à la figure 60, mais nous n'arrivons pas à expliquer que nous ne retrouvons pas le même phénomène sur la figure 60, où l'on se serait pourtant attendu à le voir amplifié.

En résumé, les résultats comparés des approches centralisées et décentralisées nous montrent l'importance de l'étape d'échantillonnage et de la densité de celui-ci : c'est cela qui définit l'espace de recherche des solutions et la qualité finale des trajectoires définies. Dans nos exemples, ce critère semble être plus important que la recherche d'une étroite coordination, qui reste de toutes façons limitée en formulation TOP à cause des hypothèses simplificatrices associées à la perception. Ainsi, du point de vue de l'oisiveté, nos approches décentralisées se montrent à la fois plus efficaces et bien plus rapides que les approches centralisées. Il reste à savoir si ce gain d'efficacité n'impacte pas négativement le non-déterminisme des plans élaborés.

8.3.2 *Non-prédictibilité*

Nous avons jusqu'ici perçu et présenté l'optimalité et la non-prédictibilité comme deux objectifs opposés. Aussi est-il légitime de s'interroger sur l'impact du gain de performances observé sur les aspects stochastiques mis en œuvre. Nous reprenons et complétons ici les résultats de la section 7.5.3 sur la non-prédictibilité des approches centralisées, en y ajoutant les résultats de nos approches décentralisées.

Le comparatif obtenu est résumé par les graphiques de la figure 62. Chacun des graphiques est associé à un des robots* et montre les distances de Fréchet moyennes entre les trajectoires du robot pour les différents environnements et pour chaque approche.

Il est important de regarder les robots un par un, car ils sont chacun associé à des conditions de tests différentes : positions de départ, topologie de l'environnement immédiat, etc.

Dans l'ensemble, les mesures de distance ne varient pas sensiblement entre les approches, hormis pour le cas notable des robots Mana et Minnie sur l'environnement de Caylus. Nous pensons que ces résultats confirment nos propos sur le compromis à trouver entre l'optimalité – la performance vis-à-vis de l'oisiveté – et la non-prédicitibilité. Il faut pour cela analyser les résultats selon le prisme de la densité d'échantillonnage : il semble alors qu'il y est un effet de seuil, atteint pour l'environnemment de Caylus et les robots Mana et Minnie ; à partir de ce seuil, la densité d'échantillonnage est telle que la résolution optimale du problème IP mène systématiquement à des trajectoires similaires. En d'autres terme, lorsque la densité d'échantillonnage est trop importante, la stochasticité tend à disparaître.

Il faut aussi prendre en compte la distance minimale entre les échantillons, bien que nous n'ayons pas déterminé avec précision son impact réel.

Ce seuil est atteint ici pour $n = 40$*– paramètre utilisé dans nos tests des approches décentralisées – pour les AGV Mana et Minnie sur l'environnement « Caylus » : la carte d'accessibilité des AGVs sur Caylus montre en effet un espace à échantillonné très étroit, comme illustré figure 32 p.121. Pour les autres configurations testées, ce phénomène de seuil de densité n'apparaît pas, l'espace accessible étant bien plus large : il faudrait pour le percevoir augmenter nettement n, ce que le solveur ne permet pas. Ce phénomène de seuil semble confirmé par l'étude des distances de Fréchet pour $(T = 100, n = 20)$ au lieu de $(T = 100, n = 40)$ – lisible sur la figure 62, mais aussi pour $(T = 200, n = 40)$: dans ce dernier cas, en divisant grossièrement par quatre* la densité d'échantillonnage, les mesures observées pour Caylus prennent des valeurs au-delà de 100, proche de ce qui est mesuré pour l'approche centralisée.

T *peut être vu comme le rayon d'action (temporel) des robots : doubler* T *revient donc grossièrement à multiplier par quatre la surface accessible.*

En résumé, les approches décentralisées sont capables de faire preuve d'autant de variabilité dans les trajectoires que les approches centralisées, mais il faut garder à l'esprit qu'augmenter la densité d'échantillonnage – ce qui est assez aisé au regard de la complexité en décentralisé – n'apporte pas seulement sur un gain de performances vis-à-vis de l'oisiveté, mais peut aussi mener à une baisse conséquente des propriétés de non-prédictibilité de nos algorithmes. Plus spécifiquement, on observe un phénomène de seuil sur la densité d'échantillonnage menant à cette chute de la stochasticité ; la valeur de ce seuil est à déterminer empiriquement.

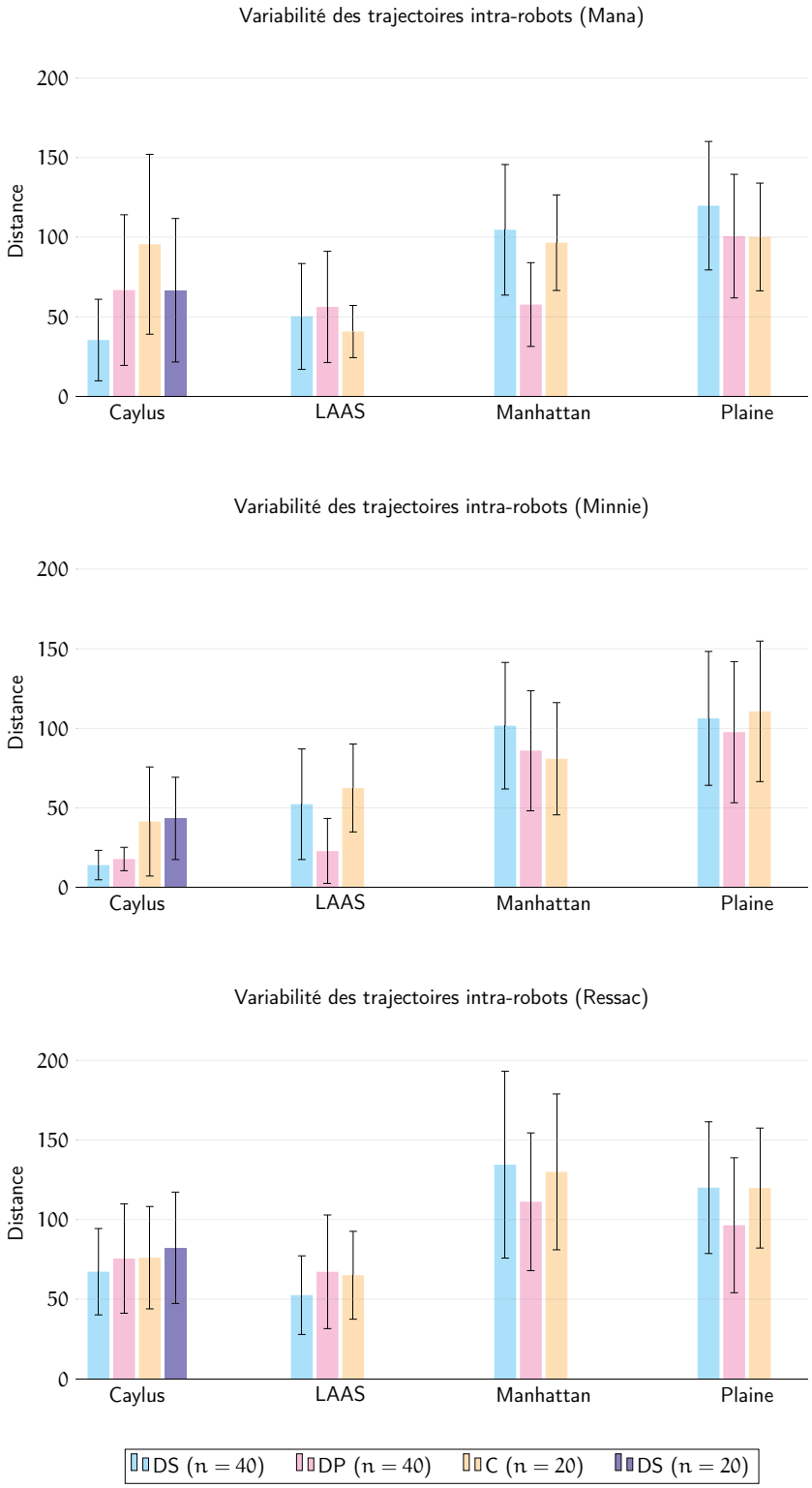


FIGURE 62 – Mesures des dissimilarités entre les trajectoires d'un même robot pour différentes instances d'un même problème de départ, avec données d'entrée et valeurs des paramètres identiques ($N = 3$, $T = 100$, $b = 3$ et $n = 20$ ou 40). Les différents graphiques permettent de comparer les approches décentralisée séquentielle (DS), décentralisée parallèle (DP) et centralisée (C), avec différentes valeurs de n. La distance indiquée est la distance de Fréchet, comme à la figure 51. Les données sont tirés des tests précédents.

8.3.3 *Formulations SP*

Jusqu'ici, la complexité inhérente aux formulations SP ne nous a pas permis de tester réellement ce type de formulations lors d'une successions de cycles. C'est regrettable car la section 7.3.4 nous a montré le potentiel de ces formulations par rapport aux formulations TOP. Au regard du gain de complexité obtenu pour ces dernières, on peut espérer obtenir un gain similaire pour les formulations SP et ainsi les rendre viables.

Malheureusement, ce raisonnement oublie de prendre en compte l'intérêt des formulations SP : leur spécificité est de distinguer les déplacements des observations. Cette spécificité est à la fois la cause de la grande complexité de ces formulations mais aussi la raison de leur intérêt. Or, elle ne fait sens que lorsqu'au moins deux robots sont en jeux : en monorobot, l'allocation des positions à observer est triviale et immédiate, le seul robot disponible héritant de toute la charge. Ainsi, il existe un nombre minimal de robots nécessaires à la justification des formulations SP : $N = 2$.

On peut alors imaginer une version semi-décentralisée dans laquelle les « grandes » équipes de robot ($N > 2$) seraient organisées en binômes, éventuellement de manière dynamique, et où chaque binôme élaborerait conjointement ses trajectoires à travers la résolution d'un problème SP. La coordination entre les binômes pourrait alors se faire comme pour les formulations TOP, en parallèle ou en séquentiel. Cette organisation en binôme peut présenter un autre avantage, en rendant chaque robot responsable d'un autre. En cas d'aléas survenant à l'un des membres de l'équipe, son binôme serait le plus à même de détecter cet aléa et d'y apporter une réponse appropriée. Malheureusement dans notre cas et avec le solveur GLPK, la complexité des formulations SP reste trop importante même pour $N = 2$, conduisant à des taux d'échecs trop élevés lors de l'étape de résolution pour permettre une application pratique et des liaisons fiables entre les instances. Nous restons donc en attente d'une intégration plus poussée exploitant un meilleur solveur afin de pouvoir tester plus en avant le potentiel de ces formulations.

8.3.4 *Aspects « très long terme »*

La plupart des tests « long terme » présentés aux chapitres 7 et 8 ont été menés sur des durées de $t \approx 2000$ – l'unité de temps restant arbitraire mais supérieure à une seconde. Nous nous posons ici la question de cette limite arbitraire : est-ce que nos algorithmes se comportent de façon attendue* au-delà de cette limite de $t \approx 2000$, dans la continuité de ce qui est observé. Nous repoussons ici la limite jusqu'à $t \approx 10000$, testée sur des séquences de cinquante instances avec le jeu de paramètre $(N = 3, T = 200, n = 40, = 3)$ en décentralisé séquentiel. Nous désignons cette limite comme le « très long terme ».

Cette section n'aborde pas ici les problèmes annexes – bien qu'important – liés au « très long terme », dont la durée des batteries, la gestion des dérives, etc.

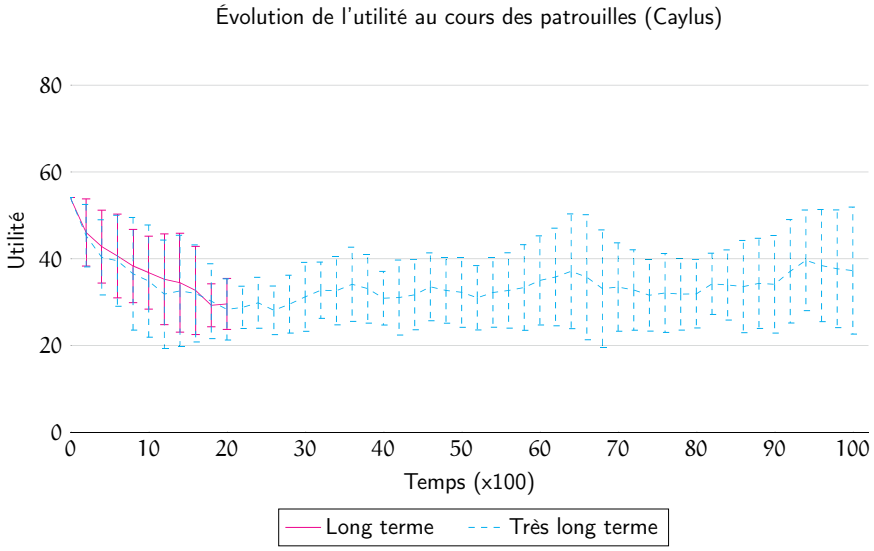


FIGURE 63 – Évolution de l'utilité moyenne à très long terme au cours d'une patrouille sur l'environnement « Caylus ». Les résultats sont une moyenne de 20 simulations ; l'écart type entre ces simulations est indiqué à chaque mesure. La courbe « long terme » reprend les résultats de la section 8.2 ; il s'agit de tests indépendant utilisant les mêmes paramètres ($T = 200$,$N = 3$,$n = 40$,$b = 3$) hormis la durée totale du test (c'est-à-dire le nombre de cycles).

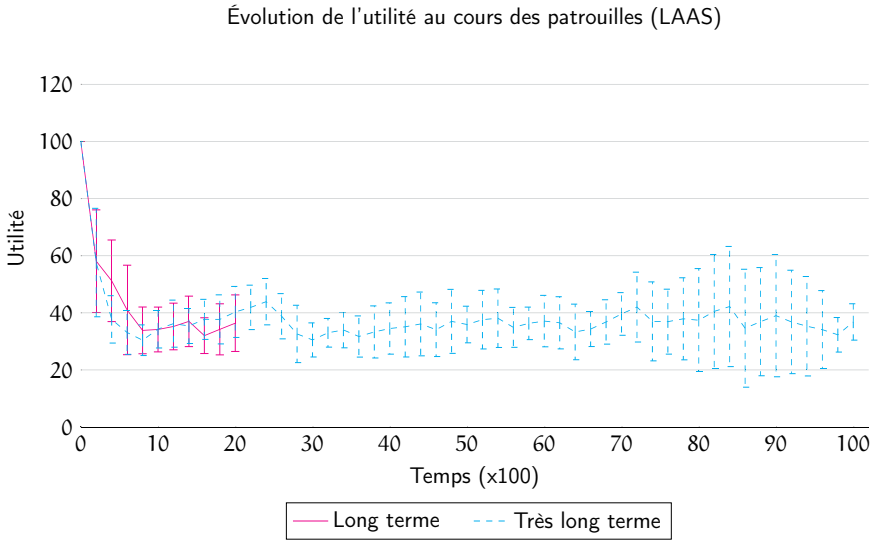


FIGURE 64 – Évolution de l'utilité moyenne à très long terme au cours d'une patrouille sur l'environnement « LAAS ». Les résultats sont présentés de la même manière qu'à la figure 63.

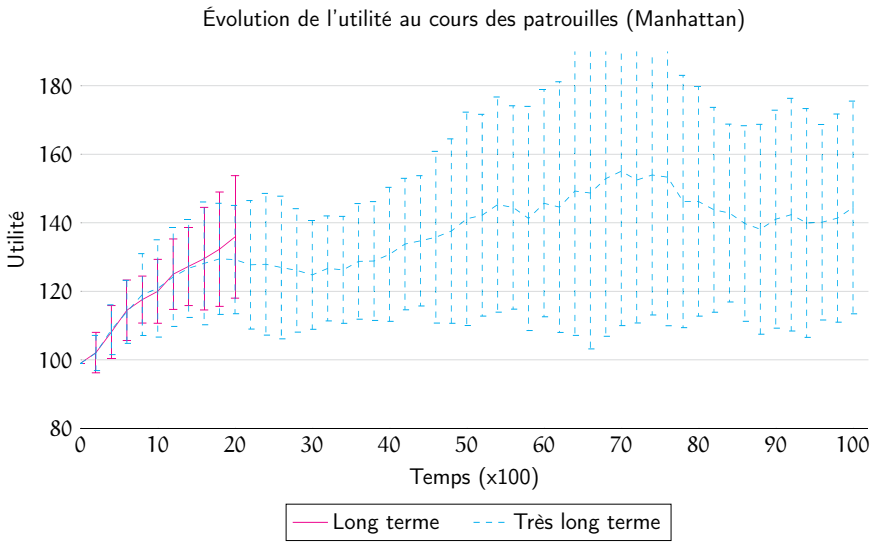


FIGURE 65 – Évolution de l'utilité moyenne à très long terme au cours d'une patrouille sur l'environnement « Manhattan ». Les résultats sont présentés de la même manière qu'à la figure 63.

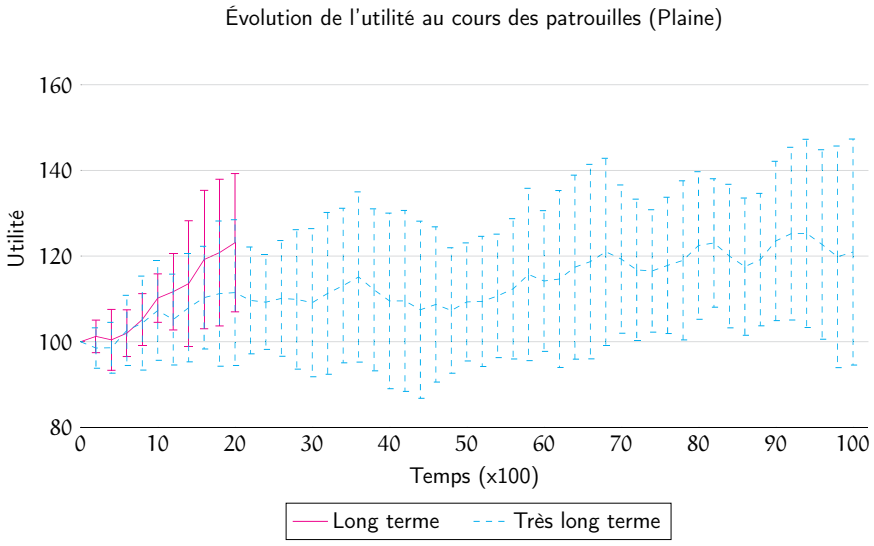


FIGURE 66 – Évolution de l'utilité moyenne à très long terme au cours d'une patrouille sur l'environnement « Plaine ». Les résultats sont présentés de la même manière qu'à la figure 63.

Les figures 63 à 66 confirment toutes dans la durée nos résultas présentés en « long terme ». En particulier, on perçoit la stabilisation durable des utilités moyennes mesurées, et de l'écart type associé. Ces résultats viennent donc avalisés la limite arbitraire retenue pour les tests en long terme.

Remarque. Nous n'avons pas indiqués ici les utilités maximales car elles ne font pas partie des fonctions-objectifs retenu pour nos problèmes IP. Néanmoins, comme attendu, nous avons pu noter qu'elles finissent par converger en moyenne, avec un phénomène d'oscillations plus ou moins marqué autour de la valeur moyenne.

8.4 COMMUNICATIONS

Nous avons présenté à la section 6.6.4 différentes pistes permettant de prendre en compte les contraintes de communications à travers les positions finales des robots. Cet aspect est important car une portée maximale de communication limitée va nécessairement contraindre les déplacements des robots ; ce phénomène est accentué par l'utilisation de modèles plus réaliste, prenant en compte les obstacles par exemple. Nous ne désirons dans notre cas que des communications périodiques, établies à la fin de chaque instance. Cela permet de relâcher un peu les contraintes portant sur les déplacements des robots, mais nous désirons néanmoins en évaluer l'impact.

Dans l'idéal, nous aimerions intégrer les contraintes de communication directement dans le calcul de la résolution des formulations IP. Cela est théoriquement possible et élégant, mais s'avère trop complexe en pratique pour être appliqué. En particulier, il est complexe de garantir que le réseau de communication établi entre les robots ne présente pas de sous-réseaux, analogue aux « sous-tours » – voir section 6.5.4. De plus, cette approche n'est pas adapté aux systèmes décentralisés, pour les mêmes raisons que les formulations SP – il ne prend de sens qu'en considérant tous les robots simultanément.

Un bon compromis pour les approches décentralisées est alors de présélectionner un ensemble de configurations finales acceptables et d'imposer au solveur de choisir parmi celles-ci lors de la résolution*. Ceci présente un double avantage : cela ne complexifie pas vraiment les formulations IP et reste valide en centralisé tout comme en décentralisé. Malheureusement, cela ne fait que déporter le problème et nécessite de pouvoir faire cette présélection. Nous pensons malgré tout que cette méthode est pertinente si un modèle de communication « simple » est disponible – « simple » au sens de « il est facile et très rapide de tester la connexion entre deux robots pour deux positions données ». Ainsi, il est par exemple facile de tester une grande quantité de configurations potentielles par échantillonnage et d'y sélectionner un nombre restreint de configurations finales correctes à fournir au solveur. Malgré notre volonté d'aller plus loin sur ce plan, nous

Avec ce type de solution, il est possible d'imposer tout type de configuration désiré – par exemple « en étoile » ou « tout connecté ». Tout dépend du module de présélection.

n'avons pas encore développé cette méthode.

Nous n'avons à l'heure actuelle développé et testé qu'une méthode dégradée permettant de prendre en compte les communications pour l'approche décentralisée séquentielle. Dans cette approche, les robots planifient l'un après l'autre ; on leur impose alors de se rattacher au réseau de communication formé par les autres robots déjà en place. Cette méthode, très simple à implémenter, pose néanmoins un problème fonctionnel : en effet, si le premier robot s'éloigne trop des autres, il est théoriquement possible que l'espace accessible aux robots suivants ne leur permette pas de se rattacher au premier, comme illustré à la figure 67.

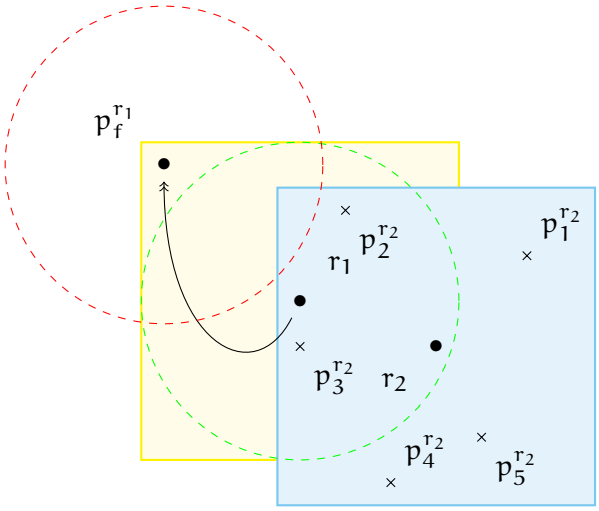
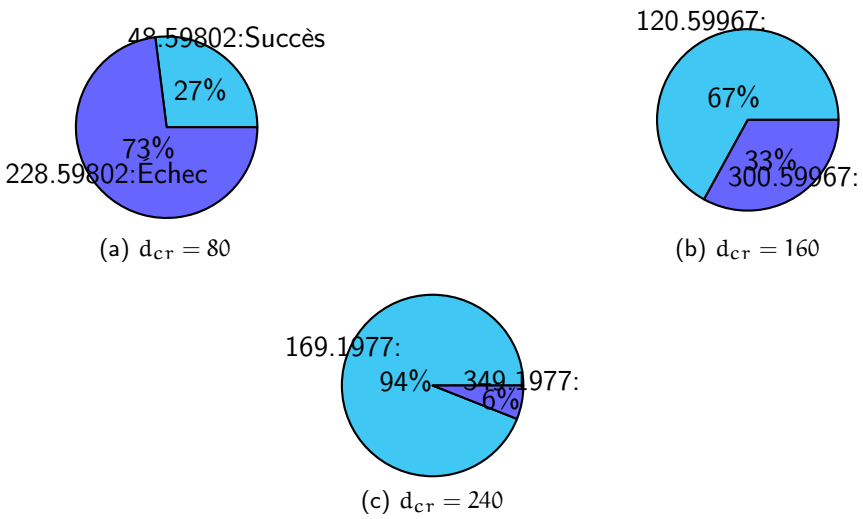


FIGURE 67 – Exemple de problème survenant lors de la méthode de traitement séquentielle des contraintes de communication : le premier robot se déplace en limite de sa zone accessible, hors de portée des positions $p_i^{r_2}$ accessibles échantillonnées pour le second robot. Cela mène à un échec du solveur (aucune solution n'existe).

.

En pratique, ce phénomène n'est pas rare, et sa fréquence dépend de la portée maximale de communication autorisée, comme illustré figure 68. Bien sûr, il est possible d'étendre cette portée mais d'une part c'est une réaction insatisfaisante et réductrice – notamment elle ne prend pas en compte les obstacles à la communication – et d'autre part on se rapproche alors de l'hypothèse de « connectivité parfaite » : la prise en compte des communications perd alors son sens. Cette méthode n'est donc pas satisfaisante. Elle permet néanmoins de montrer que les robots sont bel et bien contraints par les communications et que cela influe sur les performances observées. Par exemple, pour l'environnement « Plaine », entre $d_{cr} = 240$ (au succès récurrent) et $d_{cr} = 2000$ (= « connectivité parfaite »), on mesure un gain de performance en oisiveté croissant au cours du temps – lors de la phase de déploiement des robots – et se stabilisant autour de

8.5% au profit de l'hypothèse de « connectivité parfaite ».



(a) $d_{cr} = 80$ (b) $d_{cr} = 160$

(c) $d_{cr} = 240$

FIGURE 68 – Taux de réussite du solveur IP face à différentes portée de communication d_{cr} (*communication range*). Seule la distance est prise en compte pour établir ou non une communication. On s'aperçoit que le taux de réussite chute drastiquement avec la distance maximale de communication autorisée. Les distances d_{cr} sont à rapporter à la taille de l'environnement « Plaine » : 591×845.

En résumé, la prise en compte des contraintes communications et l'étude de leur impact forment un problème complexe. Nous avons ici mis en œuvre une solution simple, donnant des résultats médiocres et insatisfaisant. Elle a néanmoins permis de mettre en évidence l'impact significatif que peuvent avoir les contraintes de communication sur les déplacements des robots et donc sur les performances observées. Dans la discussion finale de cette thèse, nous revenons sur le problème des communications et proposons diverses considérations et solutions alternatives allant au-delà de nos seules approches du problème de patrouille.

8.5 BILAN ET PERSPECTIVES

Ce chapitre vise à évaluer l'apport de nos approches décentralisées par rapport aux résultats observés pour notre approche centralisée et présentée au chapitre précédent. Nous avons obtenu plusieurs résultats inattendus. Le résultat central est formé par les performances globales de nos approches décentralisées, meilleures que celles obtenues en centralisé. Ce résultat, surprenant de premier abord, est la suite logique de nos observations précédentes : les approches décentralisées ont une complexité plus faible, autorisant une plus grande latitude dans le choix des valeurs des paramètres. En particulier, il est possible d'augmenter nettement la densité d'échantillonnage pour améliorer l'oisiveté moyenne. Il reste tout de même

Nos résultats confirment expérimentalement nos propos du chapitre 6 sur l'échantillonnage agissant comme un curseur entre l'optimalité et la non-prédictibilité.

à trouver – empiriquement – l'équilibre souhaité entre les performances sur l'oisiveté et la non-prédictibilité des schémas de patrouille*.

De plus, notre approche décentralisée parallèle (DP) présente des performances très proches de notre approche décentralisée séquentielle (DS). Ce résultat est également surprenant car il n'existe pas de mécanismes de coordination entre les robots en DP. L'analyse des résultats a montré que la configuration spatiale – topologie de l'environnement et positions des robots – était centrale dans le besoin de coordination. En particulier, dès que les robots se sont spatialement répartis dans l'environnement, la coordination explicite des trajectoires ne présente qu'un intérêt très faible. Cette répartition spatiale, assimilable à une partition implicite de l'environnement, n'est pas expressément exprimée dans nos formulations ; pourtant, la maximisation de l'objectif – les utilités récoltées – y mène invariablement. Bien entendu, lorsque les robots sont proches les uns des autres, il reste intéressant de coordonner leurs trajectoires, bien que les formulations TOP restent limitées à cause de leur approximation des mécanismes de visibilité. D'après nos résultats sur l'impact des contraintes de communications, ces dernières pourrait accentuer le besoin de coordination en forçant les robots à se rapprocher.

Ces résultats posent alors la question de la pertinence des formulations TOP : peut-être pourrait-on leur préférer de multiples instances OP – monorobots – fonctionnant en parallèle. Elles pourraient par exemple exploiter des utilisés différentes pour chacun des agents afin de les pousser à se disperser naturellement dans l'environnement. La dispersion pourrait aussi être favorisée par l'ajout de contraintes spécifiques. Une autre alternative, exploitant les avantages de notre approche DP, serait d'optimiser les plans après coup lorsque nécessaire – voir à ce propos les travaux d'Alami sur le *plan-merging* [5].

Par ailleurs, nous rappelons ici que notre partie expérimentale s'appuie sur le développement d'un premier prototype. Les choix d'intégration – le langage Python et surtout le solveur GLPK – nous ont permis de produire rapidement des résultats intéressants montrant la pertinence et le potentiel de nos approches et formulations, et certaines de leurs limites. Malgré tout, ce prototype n'a pas été développé avec la vitesse de calcul comme objectif, et cette limitation ne nous a pas permis de mener tous les tests souhaités. En particulier, nos formulations SP sont actuellement trop complexes pour être mises en œuvre et étudiées correctement. C'est également le cas des contraintes de communications. Les solutions détériorées que nous avons proposées se sont néanmoins révélées suffisantes pour mettre en évidence l'impact négatif des contraintes de communications sur les performances mesurées ; c'est notamment le cas des contraintes portant sur la portée maximale de communication.

Le bilan global de nos travaux sur la patrouille de zones sécurisées (chapitres 6, 7 et 8) est un ensemble d'approches viables au potentiel intéressant, permettant de trouver un compromis entre l'optimalité vis-à-vis de l'oisiveté

et la non-prédictibilité des schémas de patrouille, importante en contexte antagoniste. Nous avons également vu que la décentralisation peut être une réponse supplémentaire pour aborder la complexité intrinsèque de certains problèmes, en plus du bénéfice sur la robustesse et la facilité d'intégration. Forts des résultats observés, nous souhaitons prolonger ces travaux par une intégration plus poussée de nos algorithmes, ce qui comprend notamment l'exploitation d'un solveur plus performant et du plein potentiel de la librairie *Gladys*.

9 SUIVI DE CIBLE

Un pour tous et tous pour un.

— Alexandre Dumas

Dans ce chapitre, nous proposons un nouvel algorithme de suivi de cible*– au sens de notre taxonomie, dans un contexte multirobot mais avec un principe d'économie de ressources. Ainsi, le contexte d'application de ce chapitre est très similaire à celui des chapitres précédents abordant le problème de patrouille. Les premières sections de ce chapitre présentent plus en détails le contexte d'application et formalisent le problème à l'aide des notations et des principes de modélisation précédemment introduits. La suite du chapitre explique le fonctionnement de l'algorithme et présente divers résultats expérimentaux.

Les travaux, développés tout au long de ce chapitre ont fait l'objet de deux publications [148, 149].

9.1 CONTEXTE

Comme expliqué dans notre première partie, trouver ou détecter une cible est une chose mais ne constitue en fait qu'une partie d'un scénario plus grand dans la plupart des applications réelles. En effet, il est rare que les robots puissent neutraliser la cible dès que celle-ci est détectée, et cela peut d'ailleurs ne pas être souhaitable ni attendu. Ainsi, une fois les cibles désignées, il est souvent demandé aux robots de ne pas en perdre la trace, c'est-à-dire de les pister, ce qui constitue la seconde partie des scénarios réels. Cela correspond à la branche gauche de notre taxonomie – voir figure 2 p.7. Cette dernière aborde les problèmes de pistage de cible* et y distingue plusieurs sous-objectifs : localiser précisément les cibles détectées, ou simplement les observer, chaque robot ayant à charge une ou plusieurs cibles suivant les contextes.

L'état de l'art du pistage de cible est présenté au chapitre 3.

Nous considérons dans ce chapitre un scénario multiobjectif dans lequel une équipe hétérogène de robots est en charge de contrôler une zone connue. Elle a deux objectifs concurrents : patrouiller la zone sécurisée pour détecter toute cible pouvant s'y trouver, et pister les cibles ainsi détectées, au sens du suivi de cible. La phase de détection ne prend pas fin avec la détection d'une cible : on souhaite notamment éviter que certaines cibles puissent servir de diversion au profit d'autres, par exemple lors d'une patrouille de zones sécurisées. Les robots disponibles ne sont en outre pas assez nombreux pour permettre une couverture statique de la zone : la planification des déplacements est donc centrale ici, et les tâches de suivi de cibles apparaissant en cours de mission doivent impacter aussi peu que possible le bon déroulement des patrouilles. Pour cela, elle doivent donc solliciter le

*Le temps réel souple (*soft*), par opposition au temps réel strict (*hard*), s'accommode de dépassements ponctuels des contraintes temporelles imposées, dans la limite du raisonnable (au-delà, le système devient inutilisable).*

La cible est inoffensive pour les robots au sens où ces derniers n'ont pas besoin de chercher à s'en protéger.

moins de robots possibles. Néanmoins, les cibles effectivement détectées ont la priorité sur les cibles potentielles : il est donc possible de « réquisitionner » tout robot assigné à la patrouille lorsque son intervention permet d'éviter de perdre une cible de vue. Ce processus de suivi et de réquisition a des contraintes de temps réel souple* nécessaires pour suivre la cible de manière fluide – celle-ci n'attend pas ses poursuivant.

Ce chapitre se concentre sur la phase de suivi de cible et les processus de planification associés, en respectant les contraintes soulevées par le présent contexte. La gestion du suivi se fait cible par cible : pour toute la suite de ce chapitre, sauf si cela est spécifié autrement, nous nous concentrons donc sur le suivi d'une seule cible. Nous ne faisons pas d'hypothèses spécifiques sur la cible, si ce n'est que cette dernière est supposée « inoffensive* » pour les robots. Il n'y a pas d'hypothèse sur le comportement de la cible : celle-ci peut aussi bien être un marcheur aléatoire, avoir un comportement évasif, ou encore suivre une toute autre stratégie.

Le succès de la tâche de suivi est défini par un critère de visibilité défini en termes de distance et de continuité, cette dernière allant d'une contrainte de visibilité à respecter strictement à tout instant à des contraintes plus relâchées autorisant les occultations temporaires. Conformément à notre contexte multiobjectif, nous cherchons ici à satisfaire les contraintes de visibilité tout en minimisant le nombre de robot requis pour cette tâche.

La section suivante présente notre approche et formalise le problème tout en établissant des liens avec le chapitre 5. La section 9.3 est le cœur de ce chapitre, et les algorithmes à l'origine du suivi et des réquisitions. En particulier, elle explique comment gérer la complexité intrinsèque au problème. La section 9.4 présente les résultats expérimentaux obtenus en simulation *ad hoc* et réaliste. Enfin, la section 9.5 décrit le fonctionnement des schémas d'allocation de tâches tenant lui de réquisition. Une discussion conclut le chapitre.

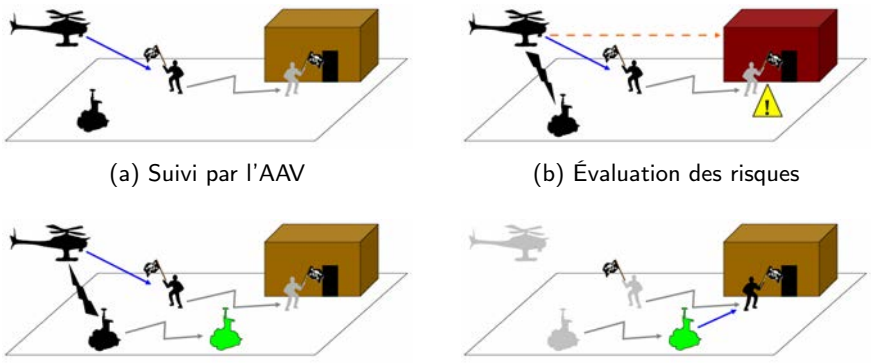
9.2 Approche : un suivi monorobot avec support multirobot

La section 3.2 de la partie I présente un état de l'art détaillé du problème de suivi, aussi appelé « poursuite-évasion » bien que ce terme soit sujet à confusion. C'est un problème qui a largement étudié depuis les premiers travaux de Eaton et Zadeh sur le sujet [50]. Les récentes contributions analytiques de Hutchinson et collab. [114, 26, 115, 25] ont montré que le problème de suivi est entièrement décidable, mais que sa complexité gène la recherche de solution optimale sous des contraintes temps réel – il est NP-complet. À ce titre, il n'est pas abordé de manière optimale dans les applications pratiques.

À l'opposé, l'état de l'art propose plusieurs stratégies locales montrant de très bonnes performances tant en simulation qu'à bord de robots, y compris dans des environnements dynamiques et obstrués [112, 19]. Il faut néanmoins noter que ces approches exploitent des modèles d'environnement pauvres : ce sont des modèles binaires 2D dans lesquels les obstacles sont communs aux mouvements et à la vision. En outre, pour aussi efficaces que soient ces méthodes, certaines situations peuvent les tenir en échec. L'idée principale soutenant notre approche est de tirer partie du contexte multirobot pour offrir un soutien au suivi, lorsque cela s'avère nécessaire pour palier les défaillances du suivi monorobot*.

Dans les cas nominaux, un seul robot s'avère amplement satisfaisant. Le cœur du problème est donc bien un problème de suivi, en un contre un.

Détaillons un peu plus cette idée : afin de minimiser les ressources nécessaires pour assurer le suivi de cible, notre approche repose principalement sur un seul robot appelé « poursuivant » afin de mener à bien la tâche de suivi : l'état de l'art montre que cela devrait être suffisant la plupart du temps. Néanmoins, à cause des capacités de la cible ou de la disposition de l'environnement, le poursuivant peut échouer dans son effort pour respecter les contraintes de visibilité, que sa stratégie soit optimale ou non. C'est pourquoi nous souhaitons que le poursuivant évalue constamment les échecs potentiels à venir et le risque associé : cela lui permet de demander des renforts lorsque cela semble nécessaire, comme illustré figure 69. Ainsi, « la cible s'apprête à entrer dans un labyrinthe » et « la cible est sur le point de traverser une zone que je ne peux traverser » sont des exemples de situations nécessitant une aide extérieure, tandis que la situation « la cible va se cacher derrière un petit bâtiment située dans une large zone sans autres obstacles » n'en n'est pas une si les occultations temporaires sont autorisées.



(a) Suivi par l'AAV (b) Évaluation des risques

(c) Demande de renforts (d) Suivi par l'AGV

FIGURE 69 – Illustration de notre approche de suivi : en (a) l'hélicoptère est en charge du suivi. En (b), la cible s'apprête à entrer dans un bâtiment dans lequel l'hélicoptère ne peut la suivre : le poursuivant demande donc des renforts (c), et l'AGV vient prendre le relais (d).

En résumé, l'état de l'art regorgeant de stratégies locales de suivi, les deux principaux problèmes soulevés par notre approche et abordés par nos travaux sont donc (i) comment prédire les échecs en cas du suivi monorobot et (ii) comment les anticiper et les prévenir en s'aidant d'un soutien multirobot.

9.2.1 *Formalisme*

Suivant notre principe d'économie des ressources, nous cherchons à minimiser les robots réquisitionnés par la tâche de suivi. Nous reprenons ici les notations introduites au chapitre 5. Nous définissons formellement le problème ainsi :

$$\min \sum_{r \in R} a_s(r,t) \qquad \forall t \in \mathcal{T} \tag{41}$$

en introduisant $t \in \mathcal{T} \subset \mathbb{R}^+$ le temps courant, et $a_s : R \times \mathcal{T} \to [0,1]$ la fonction d'activation prenant la valeur 1 lorsque le robot $r \in R$ est sollicité pour le suivi au temps t, en tant que poursuivant ou renfort.

L'équation objectif (41) doit en outre vérifier les contraintes de visibilité entre les robots et la cible :

$$\int_{t-\tau_{max}}^{t} h(R_\theta^{act}, a_{cible}, \theta) d\theta \leqslant \tau_{max} \tag{42}$$

avec $R_\theta^{act} = \{r \in R, a(r,\theta)) = 1\}$ l'ensemble des robots actifs au temps $\theta \in [t - \tau_{max}; t]$ et h définie par :

$$h(R_\theta^{act}, a_{cible}, \theta) = \begin{cases} 1 & \text{si } a_{cible} \text{ est cachée à la vue des} \\ & \text{robots } r \in R_\theta^{act} \text{ au temps } \theta \\ 0 & \text{sinon} \end{cases} \tag{43}$$

Fixer $\tau_{max} = 0$ assure des contraintes de visibilité strictes : il n'est alors pas permis de perdre la cible de vue.

En d'autres termes, la cible doit pas rester cachée plus de τ_{max} temps, qui est une durée paramètre de la mission. Nous la considérons comme une relaxation des contraintes de visibilités* : elle permet plus de souplesse dans la gestion du suivi en autorisant des occultations provisoires de la cible. Cette dernière est considérée comme cachée si aucun robot actif dans le suivi n'a de lien de perception sur elle. En reprenant la fonction de perception ϕ^r introduite au chapitre 5, on peut réécrire h de la manière suivante :

$$h(R_\theta^{act}, a_{cible}, \theta) = \max\left(0, \lfloor 1 - \sum_{r \in R} a(r,\theta)\phi^r(p^r, p^{a_{cible}}) \rfloor\right) \tag{44}$$

ce qui suppose implicitement que n'importe quel lien de perception sur la cible est suffisant pour vérifier le critère de visibilité – bien qu'il soit trivial de rajouter un seuil minimal en paramètre.

9.2.2 *Modèles réalistes*

Afin de pouvoir rendre compte d'une grande variété d'environnements de manière réaliste, et suivant nos recommandations mises en avant au chapitre 5, nous utilisons tout comme aux chapitres précédents des modèles d'environnement distinguant explicitement les capacités de déplacements (cartes de traversabilité) des capacités de perception (cartes de visibilité). Ces modèles existent pour chacun des agents, cibles incluses*.

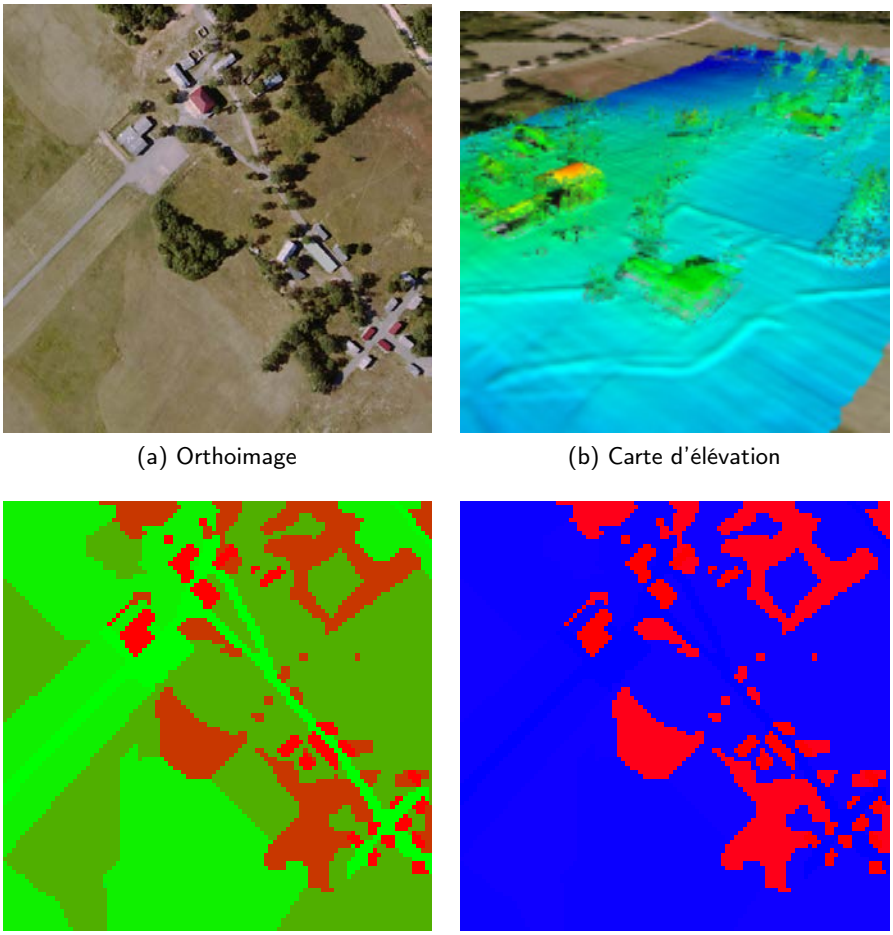
Nous supposons qu'une fois détectée, une cible est identifiée : ses capacités sont donc connues des poursuivants – mais pas nécessairement son comportement.

Nous utilisons des modèles d'environnement multicouches nous permettant de retrouver les informations d'accessibilité et de perception désirées en les convoluant aux modèles d'actions. En particulier, nous disposons d'un modèle d'élévation 2.5D et de couches sémantiques indiquant les types de terrain et leur difficulté pour les traverser (facile, difficile ou impossible), ou indiquant les capacités de visions des capteurs (luminosité). À cela s'ajoute des modèles d'actions sous forme de descriptions des capacités des agents : vitesses atteignable, capacités cinématiques, hauteur des capteurs, portée maximale, *etc.* Les capteurs des robots sont considérés comme étant omnidirectionnels pour plus de simplicité, bien que cela ne soit pas restrictif.

La convolution des modèles de déplacements et des modèles de terrain nous donne des cartes rasters d'accessibilité utilisées pour calculer les déplacements des robots et prédire le déplacement des cibles. Nous précalculons aussi des informations de visibilité *a priori* pour chaque robot (par exemple, un AAV ne peut pas voir dans un bâtiment), la carte d'élévation 2.5D venant compléter ces information pour calculer la visibilité en ligne – voir la figure 70.

9.3 Évaluer le besoin de renforts

Notre approche repose sur le fait que le robot pistant la cible, appelé « poursuivant », est capable de prédire les échecs de sa stratégie de suivi, c'est-à-dire ses pertes de vue de la cible pour une durée excédant τ_{max}. Dans cette optique, nous évaluons toutes les futures situations possibles entre la cible et son poursuivant sur un horizon temporel T, afin d'identifier quelles situations ne respectent pas les conditions de succès de la mission peuvent survenir dans l'intervalle de temps $[t, t + T]$. Ceci renvoie directement à la notion d'état introduite au chapitre 5 et à la section 5.4.3 décrivant leur manipulation. Nous décrivons ici comment les prédictions peuvent être effectuées en respectant les contraintes de temps réel souple imposées par la mission – chacun des cas d'échecs prédit génère alors une requête pour une tâche de support, dont le traitement est expliqué à la section 9.5.



(a) Orthoimage (b) Carte d'élévation

(c) Carte d'accessibilité pour les AGVs (d) Carte de visibilité pour les AGVs

FIGURE 70 – Modèles d'environnement utilisés en suivi - (a) vue satellite de la zone (orthoimage) ; (b) carte d'élévation (construite à partir d'un balayage LIDAR aérien) ; (c) carte (grille) d'accessibilité, 2.5D multicouche : les couleurs correspondent à différentes vitesses atteignables selon une échelle vert-rouge ⇔ rapide-lent ; (d) carte (grille) de visibilité 2.5D multicouche.

Remarque. Cette section se concentre donc sur la partie monorobot du suivi : nous considérons ici que l'ensemble des robots R se réduit au singleton $\{\mathrm{r_p}\}$ ne contenant que le robot poursuivant $\mathrm{r_p}$.

9.3.1 *Évaluer les échecs*

Les échecs du suivi dépendent principalement de l'environnement et des positions relatives de la cible et du poursuivant, ainsi que des états passés : il n'y a donc généralement pas de raccourci permettant de les évaluer de manière intrinsèque. Il est donc nécessaire de calculer tous les futurs états possibles sur l'horizon temporel T. Un état futur est un état d'échec si la

contrainte (42) n'est pas satisfaite, ce qui est équivalent pour le poursuivant à

$$h(\{r_p\}, a_{cible}, \theta) = 1 \qquad \forall \theta \in [t - \tau_{max}, t] \tag{45}$$

autrement dit si la cible reste cachée du poursuivant pendant τ_{max} ou plus. En utilisant (44), cela peut se simplifier ainsi :

$$\phi^r(p^{r_p}(\theta), p^{a_{cible}}(\theta)) = 0 \qquad \forall \theta \in [t - \tau_{max}, t] \tag{46}$$

Afin de calculer tous les états possibles, le temps est discrétisé, et nous exploitons la structure discrète des cartes d'accessibilité : nous définissons un arbre d'états à partir des déplacements respectifs du poursuivant et de la cible. Par soucis de simplicité, nous considérons dans cette section que la cible et le poursuivant ont des vitesses équivalentes : cela n'induit pas de perte de généricité, et influence seulement légèrement la complexité finale – voir l'annexe C pour plus de détails.

À chaque pas de temps, l'arbre d'états grossit selon un facteur de branchement de $m^{a_{cible}} * m^{r_p}$, avec $m^{a_{cible}}$ et m^{r_p} les nombres de mouvements possibles pour la cible et le robot – ceux-ci sont dénombrables à cause de la structure discrète des modèles d'environnement liés aux déplacements. Avec un horizon temporel de T, la complexité *a priori* de l'arbre d'états est de $O\left((m^{a_{cible}} * m^{r_p})^T\right)$.

$m^{a_{cible}} = m^{r_p} = 9$ (9-connexité en grille 2D) et $T \geqslant 20$ sont des valeurs usuelles pour ces paramètres. En particulier, T doit être assez élevé pour permettre aux autres robots de l'équipe de répondre aux demandes de support émises par le poursuivant, sans trop restreindre *a priori* sur leur liberté de mouvement en les contraignant à rester à proximité du poursuivant. En utilisant ces valeurs, la complexité de l'arbre est de $O(81^T)$, soit environ 10^{38} états, ce qui n'est en aucun cas gérable*. Les sections suivantes proposent différentes stratégies permettant de réduire considérablement cette complexité pour la rendre abordable en pratique.

Ce phénomène de complexité exponentielle est connu sous le nom « d'explosion combinatoire ».

Remarque. Notons que la notion d'arbre d'états n'embarque pas celle de risque d'échecs : l'arbre permet seulement de calculer ce risque. Les nœuds de l'arbre ne contiennent donc que des informations temporelles et spatiales, reliés entre eux par des liens représentant les actions (déplacements) des protagonistes.

9.3.2 *Stratégie locale de suivi*

La stratégie du poursuivant doit répondre à trois critères d'égale importance : elle doit être efficace du point de vue du suivi, c'est-à-dire permettre de garder la cible en vue autant que possible dans un maximum de situations, tout en étant prédictible – c'est essentiel pour notre approche, et très rapide à calculer afin de respecter nos contraintes de temps de calculs. En effet, nous cherchons à évaluer tous les futurs potentiels avec une cible à

la stratégie de déplacement inconnue, ce qui implique d'évaluer la stratégie de suivi du robot à de nombreuses reprises afin de gérer cet inconnu.

L'état de l'art ayant établi que calculer une stratégie optimale soulève des problèmes combinatoires, nous utilisons une stratégie locale, gloutonne. On pourrait envisager d'utiliser une stratégie optimale précalculée, mais cela ne serait pas robuste aux écarts entre les modèles d'environnement *a priori* et les environnements réels. L'état de l'art propose diverses stratégies locales temps réel efficaces, mais requiert encore des ressources de calculs non négligeable, et demande souvent de faire quelques hypothèses fortes sur le comportement des cibles, comme un comportement évasif.

Notre stratégie locale fonctionne selon les deux règles suivantes : (i) si la cible est visible, essayer de se rapprocher de la cible autant que possible tout en maintenant la visibilité ; (ii) sinon, s'engager sur le plus court chemin menant à la plus proche position permettant de satisfaire le critère de visibilité sur la cible. Cette dernière règle demande de garder une liste des positions potentiellement occupées par la cible cachée*. Notre stratégie est donc clairement sous-optimale, mais elle montre des performances de suivi correctes tout en étant prédictible et extrêmement rapide à calculer. Elle peut en outre s'appliquer à tout type de robot, AAV ou AGV.

Il possible d'exploiter un modèle probabiliste de la cible, mais cela n'est pas requis.

Cette stratégie locale présente également une propriété très intéressante : elle est déterministe, ce qui réduit le facteur de branchement de l'arbre d'états à $\mathrm{m}^{\mathrm{a}_{\mathrm{cible}}}$, le nombre de mouvements possibles pour la cible, car notre stratégie associe un et un seul mouvement de suivi en réaction à un déplacement donné de la cible. La complexité de l'arbre devient donc $\mathrm{O}\big((\mathrm{m}^{\mathrm{a}_{\mathrm{cible}}})^{\mathrm{T}}\big)$ (voir figure 71), ce qui donne environ 10^{19} en reprenant les mêmes valeurs usuelles que précédemment ($\mathrm{m}^{\mathrm{a}_{\mathrm{cible}}} = 9$ et $\mathrm{T} = 20$).

Remarque. Il est possible de réduire $\mathrm{m}^{\mathrm{a}_{\mathrm{cible}}}$ en faisant diverses hypothèses sur la cible, mais cela n'est pas désiré. En outre, le gain reste minime, la complexité restant exponentielle.

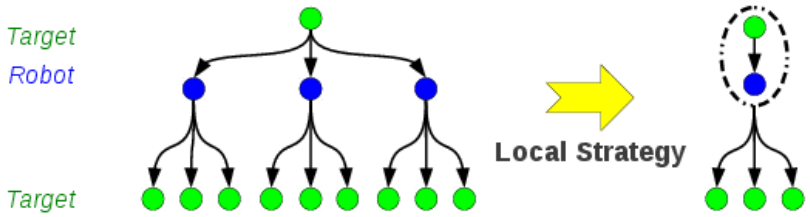


FIGURE 71 – Impact d'une stratégie locale de suivi sur la complexité (le facteur de branchement est drastiquement réduit).

9.3.3 *Arbre d'états et graphe cyclique de suivi*

Bien que significatif, le gain de complexité de la section précédente ne permet pas de satisfaire les contraintes de temps réel : il est donc nécessaire

de réduire encore cette complexité, tout en garantissant une prédiction suffisante des échecs du suivi. Nous proposons ici plusieurs transformations structurelles de l'arbre d'états permettant d'identifier et d'exploiter les redondances entre états. En effet, nous avons déjà expliqué que l'échec d'une situation de suivi dépendait d'une part de ses états passés, mais aussi et surtout des positions respectives des protagonistes. Or, le nombre de situations spatialement distinctes est fini, à cause de la structure discrète et finie des positions accessibles, construites sur des grilles. En conséquence, de nombreux états futurs possibles partagent de grande similarité spatiale qu'il est possible d'exploiter.

En premier lieu, remarquons que plusieurs positions $\mathrm{p}^{\mathrm{a}_{\mathrm{cible}}}$ de la cible peuvent être gérées depuis la même position $\mathrm{p}^{\mathrm{r}_{\mathrm{p}}}$ du robot poursuivant. Les différents états associés peuvent alors être regroupés, et les nœuds-états $(\mathrm{t}, \mathrm{p}^{\mathrm{a}_{\mathrm{cible}}}, \mathrm{p}^{\mathrm{r}_{\mathrm{p}}})$ de l'arbre d'états deviennent des nœuds de la forme $(\mathrm{t}, \{\mathrm{p}^{\mathrm{a}_{\mathrm{cible}}}\}, \mathrm{p}^{\mathrm{r}_{\mathrm{p}}})$ formant un nouvel arbre d'états, comme illustré à la figure 72.

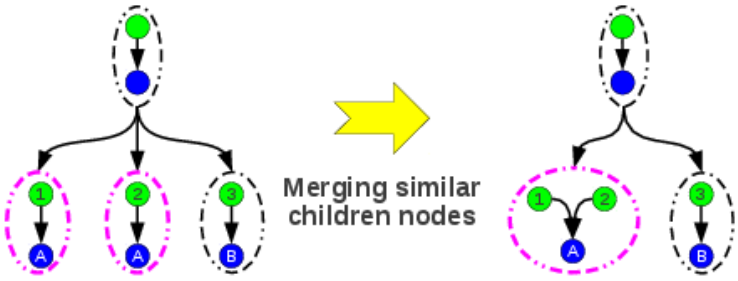


FIGURE 72 – Fusion des redondances spatiales pour réduire la complexité du graphe d'états.

En second lieu, pour un même niveau de l'arbre d'états, il existe des nœuds similaires qui ne différent que par leur parents. Or un nœud est défini par le temps qui lui est rattaché et par la disposition spatiale des protagonistes, auquel se rajoutent ensuite des liens vers les états parents (passés) et enfants (futurs). Nous proposons donc de rassembler également les nœuds similaires, quels que soient leurs parent. Cela affecte la structure de l'arbre d'états, qui se transforme en graphe. Les nœuds contiennent néanmoins les mêmes informations, comme illustré figure 73. Nous appelons ce nouveau graphe un *graphe de suivi*.

Sur un raster (grille) en 9-connexité, ces deux transformations de structure réduisent drastiquement la complexité, jusqu'à $\mathrm{O}(\mathrm{T}^5)$ dans le pire cas – voir l'annexe C pour les détails du calcul. Exploiter également les redondances temporelles permet de réduire encore d'avantage cette complexité, en introduisant des boucles temporelles dans le graphe de suivi. En effet, il est courant de revenir de revenir plusieurs fois dans une situation similaire au cours du temps, par exemple lorsqu'aucun des protagonistes

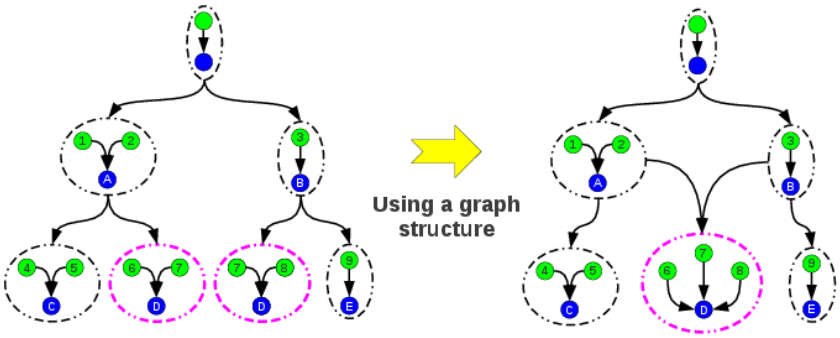


FIGURE 73 – En utilisant une structure de graphe au lieu d'un arbre, il est possible d'exploiter encore d'avantage les redondances spatiales.

ne se déplace, ou lorsqu'ils ont fait le tour d'un bâtiment. Identifier et rendre compte de ces relations temporelles dans le graphe de suivi apporte un gain supplémentaire appréciable, comme illustré figure 74. Les nœuds de ce graphe de suivi cyclique embarquent alors les données suivantes : $(\hat{t}, \{p^{a_{cible}}\}, p^{r_p})$ avec $\hat{t} = \min_t\{t/(t, \{p^{a_{cible}}\}, p^{r_p})\}$, en d'autres termes $\hat{t}$ désigne la première occurrence temporelle de la disposition spatiale considérée. On obtient alors un graphe du suivi cyclique.

Remarque. Bien que l'introduction des boucles temporelles dans le graphe induise une légère perte d'information (seules les premières occurrences temporelles sont prises en compte), cela ne s'avère pas essentiel en pratique : en effet, les tâches de support devant être gérées par ordre de priorité temporelle, seule la première occurrence importe vraiment.

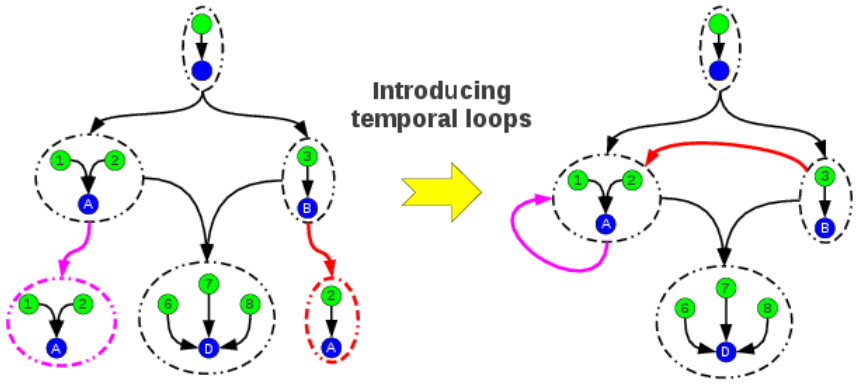


FIGURE 74 – Exploiter les boucles temporelles permet de réduire d'avantage la complexité du graphe de suivi, qui devient cyclique.

En pratique, T *est fixé de manière dynamique pour être aussi élevé que possible tout en respectant les contraintes de temps réel souple.*

Enfin, nous construisons le graphe de suivi de manière itérative, nous permettant de conserver les nœuds potentiels ayant toujours lieu d'être au pas de temps suivant, et de seulement calculer les nouveaux, c'est-à-dire de prolonger uniquement les nœuds-feuilles jusqu'au nouvel horizon temporel*. Cela ne permet pas de réduire la complexité du graphe de suivi, mais limite la complexité de calculs aux seuls nouveaux nœuds : celle-ci est alors en

$O(T^3)$ en reprenant les mêmes valeurs usuelles – se référer à l'annexe C pour le détail des calculs.

La complexité finale résultante est donc polynomiale de faible degré, ce qui nous permet d'évaluer les échecs potentiels du suivi en respectant les contraintes de temps de calcul. Par la suite, chaque échec potentiel génère une tâche de support, sous la forme (θ, p) avec θ la date de l'échec et p la position de la cible. En d'autres termes, la tâche de support est définie comme « surveiller la position p à la date θ ». Le résultat est ensemble de tâches de support, que l'on espère vide la plupart du temps – quand la cible est sous contrôle du poursuivant. Ces tâches sont les bases de la coopération multirobot, développée section 9.5. La section suivante présente, elle, les résultats expérimentaux des algorithmes présentés ici.

Remarque. Il est important de noter que toutes les transformations décrites ici mènent à une réduction importante de la complexité, sans réelle perte d'information ni d'oubli de prédiction.

9.4 Résultats expérimentaux

Cette section présente différents résultats expérimentaux* permetant d'évaluer la pertinence de l'approche et des résultats théoriques, d'abord en simulations *ad hoc* puis en simulations réalistes.

Le code source associé est intégré à la libraire LGL `http://trac.laas.fr/git/lgl` *qui entre dans le cadre du framework Jafar, maintenant obsolète.*

9.4.1 *Simulations* ad hoc

Avoir une complexité abordable et la garantie que tous les échecs potentiels seront évalués est satisfaisant sur le plan théorique, mais l'approche se révèle-t-elle pertinente en pratique ? En effet, l'intérêt de notre approche repose sur deux points importants qui n'ont pas été vérifiés jusqu'ici : d'une part, il est supposé que la plupart du temps les risques ne seront pas trop nombreux, c'est-à-dire qu'un seul robot poursuiveur sera effectivement nécessaire ; et d'autre part, il est supposé que la nouvelle complexité – polynomiale – est suffisamment basse pour permettre à nos algorithmes de tourner en temps réel. C'est ce que nous tentons de vérifier ici, à travers différentes simulations *ad hoc*.

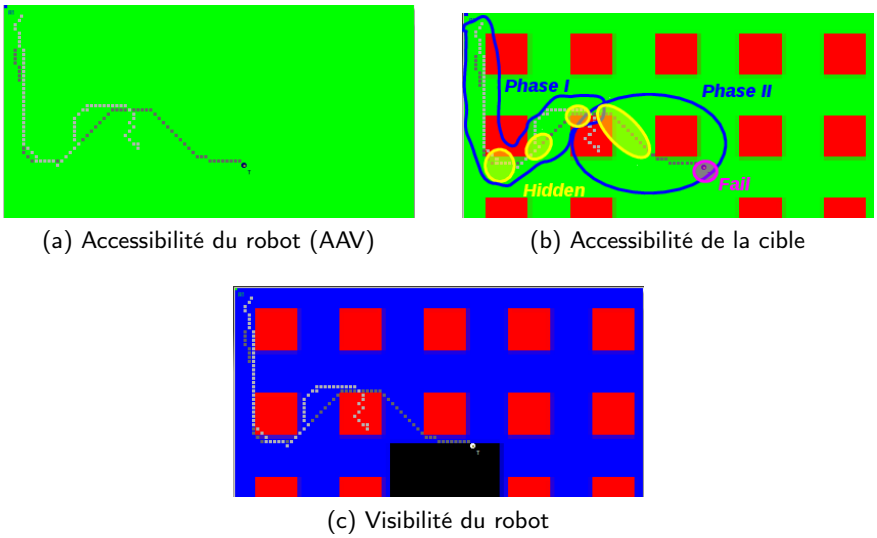
Nous présentons ici trois exemples illustrés permettant d'apprécier* les performances de la stratégie de suivi et le besoin – ou non – de support : le premier est un environnement de type Manhattan, le second une situation de « passage à gué », et la dernière correspond au terrain expérimental de Caylus, déjà présenté au chapitre 7 – voir figure 32 p. 121.

Les scénarios présentés ont été imaginés pour mettre en situation des cibles et des poursuivants aux capacités différentes : ceci montre l'intérêt des modèles riches et mène à des situations intéressantes où le poursuivant est en difficulté.

9.4.1.1 *Environnement de type « Manhattan »*

La figure 75 présente un scénario de suivi entre un AAV et une cible terrestre, dans un environnement de type « Manhattan ». Elle montre les cartes d'accessibilité et de visibilité des protagonistes. On y voit également

les trajectoires des deux protagonistes. Il est important d'y noter la présence d'une zone couverte, dans laquelle la cible peut se cacher de la vue de l'AAV quelle que soit la position d'observation de ce dernier. Nous avons découpé les trajectoires en deux phases *a posteriori*.



(a) Accessibilité du robot (AAV)

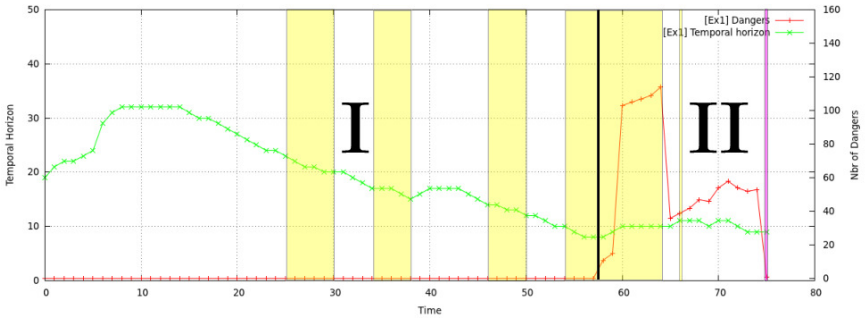
(b) Accessibilité de la cible

(c) Visibilité du robot

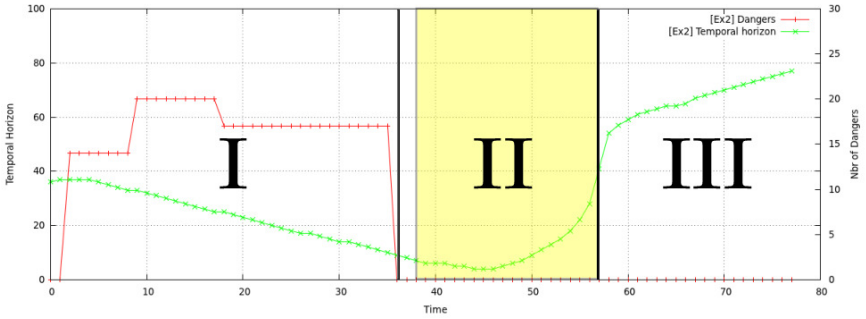
FIGURE 75 – Suivi de cible dans un environnement de type « Manhattan » : (a) carte d'accessibilité du poursuivant (un AAV) ; (b) carte d'accessibilité de la cible (terrestre) ; (c) carte de visibilité : rend compte de l'élévation des bâtiments (en rouge) et d'une zone couverte au centre (en noir) où la cible peut se cacher de l'AAV. Les trajectoires des véhicules sont dessinées en gris foncé pour la cible et en gris clair pour le poursuivant. Les zones jaunes décrivent les zones dans lesquelles la cible a été provisoirement occultée aux capteurs de l'AAV par la hauteur des bâtiments ; la zone violette (*fail*) indique l'échec la zone de fuite de la cible, où la stratégie de suivi a échoué, la cible s'étant caché dans la zone couverte. Nous distinguons deux phases dans la trajectoire.

La figure 76a montre deux courbes en rapport avec les trajectoires de la figure 75 et présente le nombre de risques (*dangers*, en rouge) évoluant au cours du temps. La courbe verte indique l'horizon temporel T : on observe une corrélation certaine entre les deux. Le nombre de risques indique le nombre d'états potentiels prédits justifiant une demande de support. Les zones en jaune correspondent aux périodes pendant lesquelles la cible était occultée. Comme indiqué précédemment, on distingue deux phases : dans la *phase I*, la cible est pleinement sous contrôle, et aucun échec potentiel n'est prédit, malgré les occultations temporaires de la cible. En *phase II*, la cible s'approche de la zone couverte, ce qui génère autant d'échecs potentiels associés. En l'absence de support, la cible finit bien par s'échapper (*fail*).

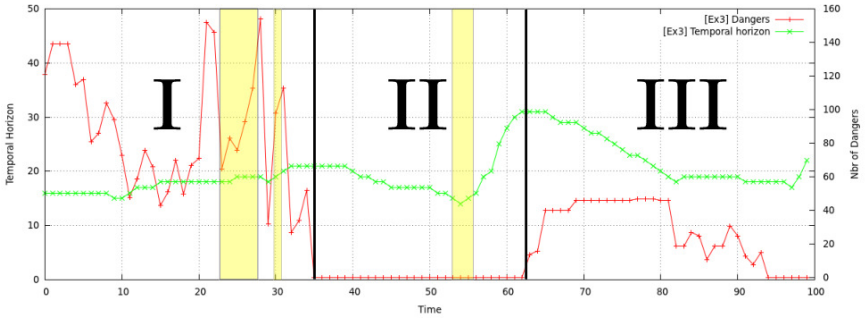
Il est intéressant de noter que l'horizon temporel T augmente au début, lorsque la situation est facile à gérer, puis diminue au fur et à mesure que les bâtiments permettent à la cible de se cacher et que celle-ci s'approche de la zone couverte. Ce phénomène illustre le dynamisme de T en fonction des contraintes de temps de calcul, T s'ajustant à la volée. C'est une consé-



(a) Environnement « Manhattan »



(b) Environnement « passage à gué »



(c) Environnement « Caylus »

FIGURE 76 – Évolution des performances de l'évaluation des risques : les courbes affichent le nombre de risques d'échecs (*dangers*, en rouge) et l'horizon temporel T (en vert) au cours du temps pour les différents scénarios présentés ici. Les zones en jaune indiquent les périodes pendant lesquelles la cible est temporairement occultée au capteur du poursuivant.

quence directe de notre stratégie à deux règles, dans laquelle les situations où la cible est temporairement occultée demandent plus de calculs que celles où la cible est visible. Il pourrait être intéressant de réviser cette stratégie pour lisser ce temps de calcul vers le bas – voir aussi la section 9.4.1.4.

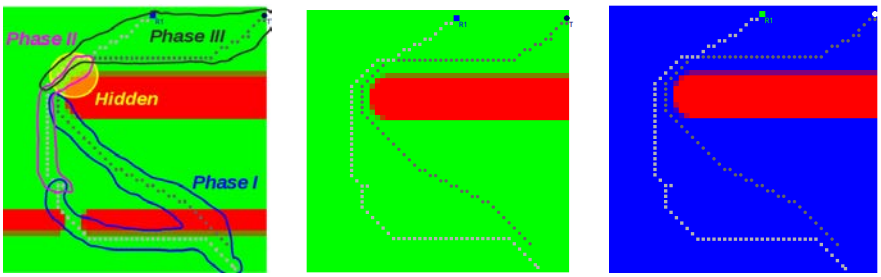
Les figures 75 et 76a soulignent les performances de notre stratégie de suivi en un contre un : bien que gloutonne et sous-optimale, celle-ci se comporte plutôt bien dans l'ensemble et exhibe des comportements intéressants. C'est le cas par exemple lorsque l'AAV se déplace le long d'un

des bâtiments plutôt que vers la cible pour garder celle-ci en vue. D'une manière générale, cela est confirmé par le faible nombre de risques d'échecs prédits – en mettant à part ceux qui sont inévitables, et par la robustesse du système aux occultations temporaires.

En outre, il est remarquable que les situations d'échecs soient regroupées temporellement et spatialement : cet aspect est essentiel car cela évite les « allées et venues » des robots réquisitionnés en renfort. On peut également imaginer que cela permet à un seul et même robot en support de répondre à plusieurs demande d'aide distinctes, aidant en cela à satisfaire notre principe d'économie des ressources – voir aussi la section 9.5. Ces propriétés ne sont pas spécifiques à cet exemple, elles se vérifient empiriquement comme illustré sur les autres exemples.

9.4.1.2 *Situation de « passage à gué »*

Nous considérons une situation mettant en scène une cible et un poursuivant terrestre, mais dans lequel la cible est amphibie, contrairement au robot. Cette situation est représentative de nombreuses autres situations dans lesquelles les cibles sont plus agiles que les robots, ou du moins ont des capacités de déplacements plus étendues. En d'autres termes, cela illustre les situations dans lesquelles le poursuivant est plus contraint que la cible. La situation présente est illustrée à la figure 77, dans laquelle la bande rouge en bas représente une rivière avec un passage à gué, le haut de la carte contenant un obstacle commun aux deux protagonistes.



(a) Accessibilité du robot (b) Accessibilité de la cible (c) Visibilité du robot

FIGURE 77 – Situation de suivi à proximité d'un passage à gué : (a) indique l'accessibilité du robot poursuivant, forcé d'emprunter le gué ; (b) indique l'accessibilité de la cible amphibie : celle-ci peut donc traverser la rivière à souhait ; (c) montre la carte de visibilité du poursuivant. La figure reprend les notations de la figure 75. Les difficultés surviennent lorsque la cible traverse la rivière, forçant le robot à s'éloigner pour emprunter le gué.

La figure 76b indique l'évolution au cours du suivi du nombre de risques d'échecs prédits et de l'horizon temporel T. Nous distinguons plusieurs phases ici aussi : la *phase I* correspond au passage à gué du robot, dans lequel il est forcé de rattraper la cible tout en la maintenant sous contrôle, si possible dans sa ligne de vue. Elle présente un certain nombre de risques que la cible s'échappe en cassant les contraintes de visibilité – occultation

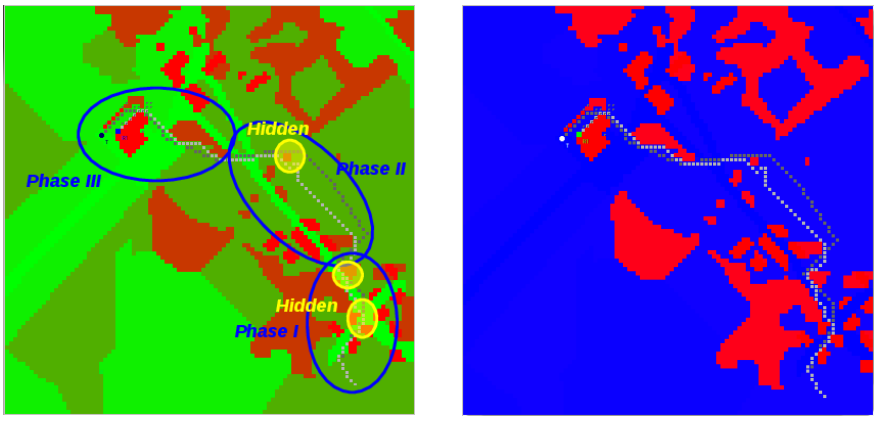
trop prolongée. La trajectoire de la cible n'étant pas une trajectoire de fuite optimale, les risques finissent par disparaître, ce qui correspond à la *phase II*. Pendant cette phase, la cible réussit à se cacher de son poursuivant, mais cela est évalué comme temporaire et ne comporte donc pas de risque. Dans la *phase III*, le poursuivant a rattrapé la cible et se trouve dans une configuration nominale constituée d'un espace libre d'obstacle. L'environnement constituant même un cul-de-sac facile à gérer en suivi, l'horizon temporel atteint des valeurs élevées – il dépasse les 60 pas de temps.

9.4.1.3 *Environnement « Caylus »*

Le site de Caylus est le même que celui utilisé aux chapitres 7 et 8, mais dans une zone différente – d'où les cartes d'accessibilité sensiblement différentes.

Ce troisième exemple est tiré d'un environnement réel : le site expérimental de Caylus. La cible et son poursuivant sont considérés comme partageant la même accessibilité, et l'environnement présente plusieurs zones encombrées d'obstacles dans lesquelles assurer un suivi est difficile. Il y a également quelques zones étendues sans obstacles – voir figure 78.

La figure 76c permet d'identifier clairement les zones à risques : elles correspondent comme attendu aux zones encombrées d'obstacles. Dans les zones plus simples, la stratégie gloutonne de suivi suffit à prévenir tout risque d'échecs. Soulignons tout de même que cette stratégie se montre pertinente également en zone encombrée, malgré les risques d'échecs : elle assure un horizon temporel correct et un nombre de dangers limité malgré tout. En outre, si elle ne peut prévenir la fuite d'une cible évasive suivant une trajectoire optimale, elle se montre tout à fait capable de suivre une cible évasive sous-optimale.



(a) Accessibilité du robot et de la cible (b) Visibilité du robot

FIGURE 78 – Situation de suivi dans en environnement réel encombré (le site expérimental de Caylus) : (a) l'AGV poursuivant et la cible partagent la même accessibilité ; (b) la carte du visibilité du poursuivant. La figure reprend les notations de la figure 75. Les difficultés surviennent dans les zones encombrées d'obstacles (*phase I* et *phase III*).

9.4.1.4 *Performances temps-réel*

Nous avons établi précédemment que la borne supérieure de complexité pour la construction du graphe de poursuite dans le cas d'un environnement en grille en 9-connexité était de $O(T^3)$ au pire cas – ce nombre est une borne supérieure correspondant à un environnement sans obstacle. La figure 79 le vérifie empiriquement pour l'environnement de Caylus, avec un très bon coefficient de détermination R^{2*}. En fait, dans ce cas où l'environnement contient de nombreux obstacles, le graphe de poursuite est même de taille plus réduite – il y a moins de configurations possibles car moins de positions accessibles possibles.

Le coefficient de détermination R^2 permet d'évaluer la qualité d'une régression : plus il est proche de 1, meilleure est la régression.

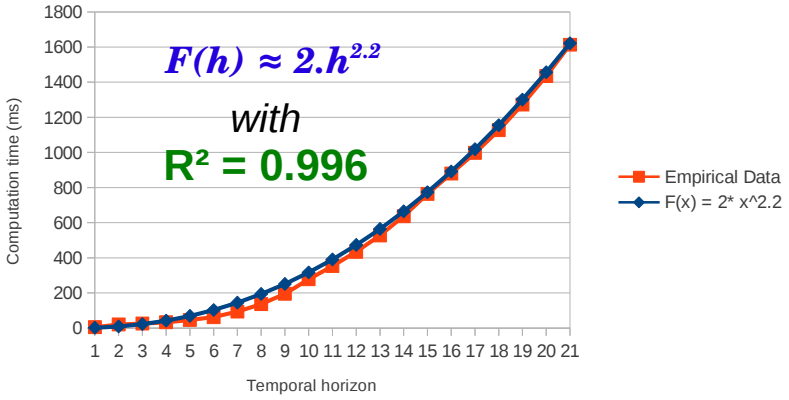


FIGURE 79 – Évolution typique du temps de calcul en fonction de l'horizon temporel de planification du suivi, comportant le calcul de la trajectoire et l'évaluation des risques associés. La régression polynomiale donne une complexité presque quadratique.

Cette complexité permet des performances en temps-réel souple comme souhaité. Nous y associons une limite de temps de calcul située à environ 1 seconde ce qui correspond grossièrement à un pas de temps, voire même à un demi-pas de temps. Ainsi, tout en maintenant un horizon temporel raisonnable (au dessus de 15), cette limite est ainsi globalement respectée, comme illustré à la figure 80. Comme nous l'avons déjà dit, l'horizon temporel s'adapte dynamiquement au temps de calcul, celui-ci dépendant de l'horizon temporel actuel – le coût augmentant avec l'horizon – et de la configuration du terrain – obstacles au mouvement ou à la vision.

Remarque. On peut remarquer sur la figure 80 que le temps de calcul du graphe de poursuite excède parfois la limite virtuelle de « une seconde » choisie pour définir notre temps réel souple. Cela s'explique en fait par la manière dont le système calcule actuellement les états potentiels : il les calcule horizon temporel par horizon temporel – et non un par un. L'estimation du temps de calcul pour étendre entièrement de 1 l'horizon temporel n'étant pas très précise, il arrive que le système choisisse d'étendre l'horizon temporel alors qu'il n'en a pas tout à fait le temps, et excède ainsi

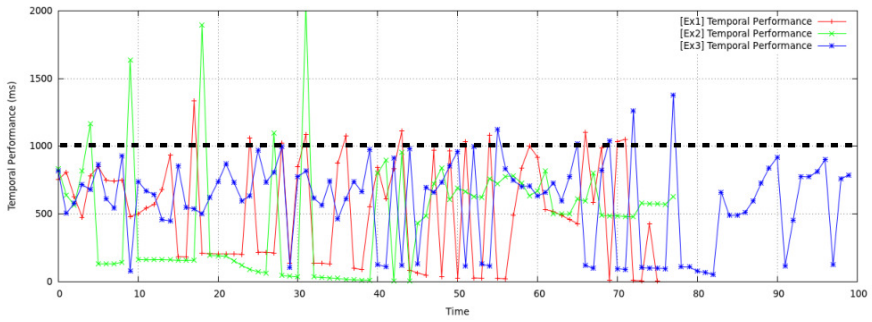


FIGURE 80 – Cycle de calculs du graphe de poursuite à chaque pas de temps, pour chacun des trois exemples précédents. La limite en pointillée représente la limite virtuelle de « une seconde ».

la limite de temps de calcul. Nous contrebalançons cela avec le cycle de calcul suivant, qui est toujours très court. En calculant les états un à un, cette limite serait strictement respectée – à quelques millisecondes près – et aurait pour effet bénéfique de « lisser » l'horizon temporel moyen, voire de l'augmenter – il serait possible de calculer plus d'états potentiels à chaque étape.

9.4.2 *Simulations réalistes*

Bien que les modèles d'environnement considérés dans nos simulations *ad hoc* soient réalistes, ils ne peuvent rendre compte de tous les problèmes cachés apportés par l'étape d'intégration. La plupart de ces problèmes peuvent être mis en évidence par l'utilisation de simulateurs réalistes, comme discuté à la section 4.3 de la partie I. Dans cette section, nous développons les résultats obtenus lors de l'intégration de nos algorithmes pour une utilisation expérimentale exploitant le simulateur Morse [51].

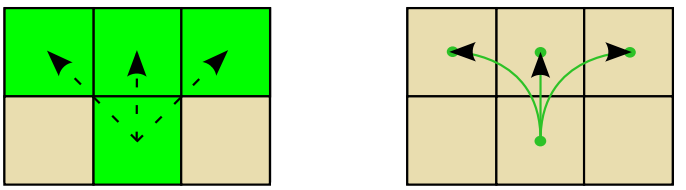
Ces tests avancés nous permettent de mettre en lumière les limites de notre implémentation et de certaines hypothèses, sans remettre en question les principes au cœur de nos algorithmes. En effet, ces limitations proviennent d'un écart parfois trop important entre la réalité et les modèles, ce qui est problématique comme développé précédemment – voir la figure 22 p.77. Cet aspect touche particulièrement notre modélisation du mouvement.

Ainsi, rappelons que jusqu'ici nous avons utilisé une grille à base de carrés pour discrétiser l'environnement. Cette étape de discrétisation est essentielle pour le fonctionnement de nos algorithmes sur l'arbre d'états et le graphe de poursuite, car c'est le caractère discret et redondant des positions qui permet de gérer la complexité sous-jacente. Pourtant, un environnement réel étant par nature continu (dans son ensemble), de même que les mouvements des agents à l'intérieur, la discrétisation mène inévitablement à des cassures et des effets de bord, que les simulations réalistes (qui présentent un temps, un environnement et des mouvements continus) mettent en évidence.

Ces dernières font apparaître plusieurs problèmes. Le premier vient de la difficulté qu'ont certains robots non holonomes à atteindre des positions précises. Ce phénomène touche particulièrement les AGVs, en général moins agiles que les AAVs, et est accentué lorsque la position d'arrivée est proche de la position de départ – le mouvement global étant alors plus contraint. Pourtant, il nous est nécessaire d'avoir une certaine finesse dans la granularité de nos modèles d'environnement : c'est elle qui nous permet de rendre compte de ses spécificités. Il nous faut donc trouver un compromis, peu évident à déterminer.

De plus, lorsque les positions sont difficiles à atteindre, le robot poursuivant peut passer plus de temps qu'attendu à effectuer son mouvement (à cause de manœuvres supplémentaires), excédant alors le pas de temps de la discrétisation temporelle. On se retrouve alors avec un décalage entre l'état réel du monde et la représentation interne, constituée du graphe de poursuite et des états prédits. Cela peut mener à des échecs non gérés par le système, car non évalués.

Heureusement, des solutions existent. En effet, l'implémentation de notre système utilise des estimations de coûts de mouvement que ne sont pas compatibles avec des véhicules non holonomes peu agiles, mais le sont par exemple avec des AAVs de type quadrirotor. En outre, pour les plateformes non holonomes (AGVs ou AAVs à voilure fixe), il nous faut revoir la représentation de l'environnement, et l'ancrer plus fortement sur les mouvements possibles du poursuivant. Le graphe de poursuite n'est alors plus construit sur une grille, mais à partir d'un autre ensemble fini de positions discrètes proposant une nouvelle partition de l'environnement. Pour construire cet ensemble, nous proposons un échantillonnage des mouvements possibles du poursuivant, comme illustré par la figure 81. Notons qu'il est impératif d'ajouter une nouvelle dimension à l'espace en considérant l'orientation des robots.



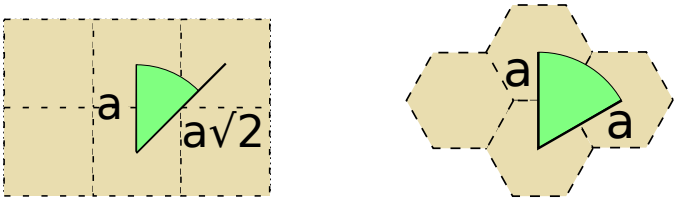
(a) Discrétisation par grille carrée

(b) Discrétisation basée sur le mouvement

FIGURE 81 – Discrétisation de l'espace : (a) à partir d'une grille carrée ; (b) à partir d'un échantillonnage des mouvements possibles. Cette seconde méthode est plus cohérente avec la réalité.

Par ailleurs, les plateformes holonomes sont elles aussi sensibles à la discrétisation de l'espace : l'utilisation d'une grille carrée en 9-connexité comme canevas de discrétisation génère des chemins de longueurs variables entre les positions comme illustré à la figure 82, ce qui est gênant dans notre contexte de synchronisation étroite entre le temps et l'espace. Il est

possible de considérer d'autres types de grilles ne présentant pas ce problèmes, par exemple les grilles hexagonales. Une discrétisation basée sur le mouvement ne comporte naturellement pas ce problème de chemins de longueurs différentes.



(a) Grille carrée (b) Grille hexagonale

FIGURE 82 – Différents types de grilles donnent des longueurs d'arête différents dans le graphe de poursuite : (a) avec une grille carrée, les longueurs sont hétérogènes ; (b) avec une grille hexagonale, les longueurs sont homogènes.

Tous ces changements dans la discrétisation de l'espace et la représentations des mouvements possibles affectent la taille et la topologie du graphe de poursuite Néanmoins, ils ne changent pas fondamentalement la complexité : seuls les coefficients polynomiaux sont affectés – se référer à l'annexe C pour revoir les détails du calcul de complexité. Notre approche et les algorithmes la constituant ne sont donc pas remis en cause.

9.5 SUIVI ET RENFORTS

Tout en assurant sa tâche de suivi, le poursuivant évalue tous ses échecs potentiels jusqu'à l'horizon temporel $t + T$, avec t le temps courant. Ces prédictions indiquent les besoins de renfort, et génèrent directement les tâches supports associées sous la forme « observer la position p au temps θ », avec p une des positions discrètes retenues et $\theta \in [t, t + T]$. Par construction, le poursuivant à l'origine de ces tâches de support ne peut les gérer lui-même. Il en garde néanmoins la gestion : c'est-à-dire que c'est lui qui décide qui en aura la charge, parmi ses coéquipiers pouvant les mener à bien. Cette allocation de tâches se fait selon la disponibilité des autres robots, leurs capacités, priorités, et l'objectif global de minimisation du nombre de robots sollicités par la poursuite – voir l'équation (41). Cette section discute de ce processus d'allocation.

Seuls des travaux préliminaires ont été menés à ce sujet, aussi cette section ne présente pas de résultats expérimentaux détaillés, mais nos premiers tests ont montré des résultats prometteurs avec de très bonnes performances de suivi impliquant un nombre limité de robots seulement, tout en respectant les contraintes temps réel. Les paragraphes suivants développent les problèmes soulevés par l'allocation des tâches de support et les solutions retenues.

Remarque. Les problèmes de communication ne sont pas considérées ici ; elles impactent essentiellement la connaissance des robots vis-à-vis de la

situation, et ce qui peut mener à des incohérences à ce niveau au sein de l'équipe, mais elles peuvent aussi gêner le poursuivant dans sa recherche de soutien.

L'obsolescence d'une tâche de support peut libérer un robot en soutien avant que celui-ci n'ait mené sa tâche à bien, celle-ci n'ayant plus de raison d'être.

L'ensemble des tâches de support est mis à jour à chaque pas de temps : suivant les mouvements de la cible, de nouvelles tâches apparaissent et d'anciennes tâches deviennent obsolètes*. Le poursuivant diffuse ces mises à jour aux autres robots, qui à leur tour mettent à jour leur liste interne de tâches, et évaluent s'ils peuvent fournir le soutien demandé par les tâches non allouées : nous faisons alors face à un problème d'allocation de tâche tout à fait classique.

Les problèmes d'allocation de tâches ont fait l'objet de nombreux travaux, dont certains ont été cités dans la première partie de la thèse. Ils soulèvent des problèmes combinatoires et trouver une solution optimale reste très complexe pour des problèmes de tailles raisonnables. Néanmoins, de nombreuses solutions ont été proposées, par exemple à base de méthodes stochastiques [159] ou de mécanismes d'enchères (se référer à [47] pour une étude plus poussée). Ces dernières en particulier se sont révélées efficaces pour fournir des solutions correctes. Choi et collab. [39] présentent par exemple un algorithme menant à des bonnes solutions avec certaines garanties d'optimalité. Les auteurs y soulignent la difficulté des problèmes combinatoires soulevés par la construction des lots de tâches* (*bundles of tasks*), et notamment les conflits soulevés par une construction itérative de ces lots.

Un lot de tâches est un sous-ensemble de tâches attribuées à un même agent. La construction de lots pertinents est complexe.

ALLOCATIONS DE TÂCHES Nous avons choisi d'utiliser des mécanismes d'allocation par enchères, car ces approches ont montré de bons résultats et peuvent gérer facilement les incohérences éventuelles entre les connaissances situationnelles des robots. Néanmoins, nous gérons des coûts assez différents de ce qui est rencontré habituellement : en effet, notre objectif étant de minimiser le nombre de robots impliqués, effectuer une ou plusieurs tâches à le même coût si elles impliquent le même nombre de robots, alors que la récompense est elle bien linéaire avec le nombre de tâches effectuées.

Nous avons choisi de faire des enchères à un tour, où plusieurs offres sont autorisées : après avoir mis à jour sa liste de tâche, chaque robot calcule plusieurs lots de tâches qu'il est capable d'effectuer, et les proposent en retour au commissaire-priseur (le robot poursuivant). Ce dernier détermine alors la meilleure combinaison de lots selon l'équation-objectif (41) tout en respectant les contraintes de visibilités, c'est-à-dire en couvrant tous les cas d'échecs possibles (chaque tâche de support doit être attribuée). Les tâches sont finalement allouées par sous-ensembles (soit un lot entier, soit une partie d'un lot dans le cas de redondances entre les lots).

Trouver la combinaison optimale de lots est coûteuse : en considérant N_t le nombre moyen de lots pour chaque robot et en notant N le nombre de robots enchérisseurs, il y a $O({N_t}^N)$ combinaisons possibles. Des valeurs

usuelles sont $N_t = 3$ et $N = 4$, soit $4^4 = 256$ combinaisons possibles. Cela reste raisonnablement bas, et il est tout à fait envisageable d'évaluer exhaustivement les différentes combinaisons de lots. Ces dernières sont ensuite triées selon le nombre de tâches effectuées (*max*) puis le nombre de robots impliqués (*min*). Le commissaire-priseur sélectionne enfin la meilleure et en informe les robots impliqués dans les enchères.

CALCUL DES LOTS DE TÂCHES Les robots en soutien calculent les lots de tâches qu'ils peuvent effectuer selon leurs capacités, leurs priorités, et leurs précédents engagements. Ils ne peuvent être exhaustifs dans l'élaboration de ces lots, car n tâches donnent $\sum_{p=1}^{n} \binom{n}{p} = 2^n - 1$ lots possibles. C'est pourquoi nous avons choisi de ne ne calculer que les plus gros lots, et ce de manière itérative. Nous voulons aussi que chaque tâche soit rattachées à au moins un lot, dans la limite des capacités du robots. Au final, les lots calculés forment une partition redondante de l'ensemble des tâches de supports émises.

Nous avons remarqué dans la section 9.3 que les tâches de support générées avaient pour propriété d'être regroupées spatialement et temporellement : cela permet de ne former et garder qu'un petit nombre de lots de tâches, habituellement une poignée au plus. Les tâches étant indépendantes les unes des autres, il est possible de les regrouper ou de les séparer à loisir.

Remarque. Nous avons délibérément déplacé la lourde charge de calcul de la phase d'enchères de l'allocation vers la construction des lots de tâches : ceci économise les ressources en calcul du robot commissaire-priseur afin qu'il les concentre pour le suivi – il est aussi poursuivant. Cela lui permet d'étendre au mieux le graphe de poursuite, qui est au cœur de notre approche. Pour les mêmes raisons, nous avons introduit un décalage temporel d'un pas de temps entre le moment où les tâches de support sont émises et celui où les enchères associées sont rassemblées : cela permet en quelque sorte de paralléliser les processus de construction du graphe et d'allocation des tâches, dans le but de respecter nos contraintes de temps réel souple tout en maintenant un horizon temporel suffisamment élevé pour être pertinent. Le léger décalage introduit reste négligeable, au sens où $1 \ll T$.

9.6 BILAN ET PERSPECTIVES

Ce chapitre présente une approche originale pour le problème de suivi de cible dans des conditions réalistes, mettant en œuvre plusieurs de nos recommandations faites dans la partie I et au chapitre 5. Considérant qu'une équipe de robot est général en charge de plusieurs objectifs distincts, notamment de détection et de suivi, nous fournissons une schéma de coopération centré sur un suivi monorobot, dans lequel le poursuivant évalue en permanence son risque d'échecs et demande du renfort en cas de besoin, suivant un principe d'économie de moyens. Nous exploitons des modèles d'environnements réalistes capables de rendre compte d'une large variété

Les travaux présentés dans ce chapitre font partie des premiers à exploiter des modèles réalistes en suivi de cible.

d'environnement, ainsi que des différences de capacités entre les cibles et les robots. Notre approche va donc bien au-delà des modèles traditionnels construits sur des cartes 2D binaires et monocouches*.

Notre validation expérimentale confirme notre analyse théorique de la complexité : bien que notre approche soit intrinsèquement complexe, une étude approfondie des redondances des configurations de suivi possibles permet de rendre le problème abordable en temps réel souple. La validation expérimentale, bien que manquant de considérations statistiques, montre que notre approche répond plutôt bien à notre principe d'économie de ressources, en ne sollicitant que peu de robots, et un seul la plupart du temps. Par ailleurs, un début d'intégration a mis en lumière l'importance de la topologie des modèles d'environnement discrets : la grille cartésienne carrée, bien que très pratique car facile à gérer informatiquement, présente de nombreuses limites et s'avère peut adaptée à des modèles réalistes : on lui préfèrera d'autres types de découpage, éventuellement basés sur le mouvement du poursuivant.

La principale limite actuelle de notre approche est liée aux communications, dont les difficultés associées ont pour l'instant été mises de côté. En revanche, l'approche est plutôt robuste aux échecs dans la réalisation des plans calculés, car elle fonctionne en ligne et peut donc s'adapter assez facilement à tout aléa.

Les premiers développements futurs envisagés, outre une intégration plus poussée basée sur découpage de l'environnement selon les mouvements des robots, portent sur une résolution ajustable de la discrétisation spatiale et temporelle dans le but de mieux anticiper les échecs, et dans l'exploitation de modèles probabilistes du comportement de la cible. Ces derniers s'intègrent parfaitement dans le cadre théorique de notre approche, mais ces modèles étant difficiles à définir avec précision, nous souhaitons étudier l'impact de modèles inexacts sur les performances des algorithmes. Des considérations plus générales, dont un rapprochement avec nos algorithmes de patrouilles, sont discutées au chapitre suivant.

DISCUSSION

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth

Le présent chapitre reprend et résume les contributions principales constituant le cœur de notre thèse, et présente les développements futurs que nous aimerions y apporter. Il discute ensuite des perspectives de recherche ouvertes ou renforcées par nos travaux, ce qui conclut ce document.

RÉSUMÉ DES CONTRIBUTIONS

Dans la première partie de notre thèse, nous avons développé un ensemble de considérations sur la planification en robotique mobile, avec pour cadre applicatif les scénarios de détection et suivi de cibles. Ce champ d'application étant vaste et largement abordé par différentes communautés scientifiques, nous avons proposé une taxonomie orientée problème permettant de rassembler et comparer les travaux relatifs à des problèmes similaires. Nous avons alors développé un vaste état de l'art, centré sur les approches multi-robots mais cherchant à aller au-delà des frontières de chaque communauté. De l'analyse de cette état de l'art, nous tirons en bilan qui se résume en trois points qui sont, selon nous, autant de directions actuelles et futures où porter les efforts de recherche : (i) l'exploitation des modèles plus riches, (ii) le développement d'algorithmes décentralisés, et (iii) l'amélioration des processus de validation expérimentale.

Dans en second temps, nous nous sommes appuyés sur ces considérations pour proposer un ensemble cohérent de propositions algorithmiques. Nous avons tout d'abord rappelé l'importance des modèles pour la planification en robotique : ils lui sont nécessaires, et nous les voyons comme le pont reliant les problèmes concrets abordés par la robotique et les solutions abstraites apportées par l'intelligence artificielle. Nous proposons un organisation spécifique des modèles permettant l'exploitation cohérente de modèles riches de manière conservative, c'est-à-dire permettant une transformation retour des solutions abstraites en des solutions concrètes, ancrées dans la réalité. Cette organisation s'appuie sur une séparation claire des modèles et des algorithmes. Nous avons ensuite mis œuvre cette organisation et les principes associés pour des problèmes de patrouille de zones sécurisées et de suivi de cibles.

Nous avons développé une solution stochastique aux problèmes de patrouille en contexte antagoniste à travers la notion d'instance : une instance

est un sous-problème construit par échantillonnage à partir du problème d'origine. L'étape d'instanciation – l'échantillonnage – permet de gérer la complexité et forme un compromis ajustable entre l'optimalité et le non-déterminisme des solutions. Nous avons proposé une validation rigoureuse de nos formulations et résultats théoriques, et développé deux extensions décentralisées de notre approche par instanciation. Ces approches décentralisés se sont montrées particulièrement efficaces à travers le gain de complexité obtenu. Elles ont également permis de questionner les besoins de coordination entre les robots. Nous avons conclu cette partie algorithmique en proposant une solution hybride au problème de suivi de cible, à mi-chemin entre le mono- et le multirobot, en appliquant un principe d'économie des ressources. Cette approche a été conçue avec l'idée de mener à bien des missions de détection et de suivi de cibles en parallèle plutôt qu'en séquence.

PERSPECTIVES À COURT TERME

Lors de nos travaux, nous avons fait et développé de nombreux choix. Nous aimerions revenir sur certains d'entre eux, pour les compléter et les comparer avec des alternatives laissées de côté, voire pour les changer au regard des résultats expérimentaux et de l'expérience acquise. Nous présentons ici les prolongements que nous aimerions apporter dans la suite logique de nos travaux et des résultats obtenus. Chacun d'entre eux représente un travail à court terme allant de quelques semaines à quelques mois.

Le premier prolongement de nos travaux est assez immédiat, tant dans son choix que le temps requis pour le développer : il s'agit d'intégrer effectivement *Gladys* avec nos algorithmes de patrouilles et de suivi. En effet, nous en utilisons actuellement des versions détériorées : l'« ancêtre » de *Gladys* pour le suivi, et une émulation de *Gladys* en Python pour la patrouille. Cette intégration demande un peu de temps mais ne pose pas de difficultés techniques particulières. Cette intégration apporterait une meilleure gestion des modèles – par exemple des évaluations de coûts plus précises.

Un second développement important porte sur l'intégration d'un solveur IP plus performant pour nos algorithmes de patrouilles. En effet, le solveur retenu – GPLK – a montré ses limites sur les temps de calcul mesurés et face aux formulations SP, que nous n'avons du coup pas réellement pu mettre en œuvre. Nous espérons qu'utiliser d'autres solveurs plus performants apportera une réponse satisfaisante à ces problèmes. En particulier, certains solveurs commerciaux proposent des licences gratuites aux académiques, tout en montrant des performances bien au delà de celles de GLPK – voir annexe B. L'intégration d'un nouveau solveur demande du temps – notamment pour apprivoiser le nouveau solveur choisi – mais ne devrait pas poser de difficulté technique particulière. Les bénéfices attendus sont doubles : à performances égales, réduire les temps de calculs mesurés pour les formulations TOP d'un à deux ordres de grandeur ; et permettre l'étude approfondie des formulations SP. En particulier, on souhaite évaluer le gain

de performances attendu par rapport aux formulations TOP, et discuter de son intérêt face au coût de complexité et donc aux contraintes induites sur les jeux de paramètres acceptables.

De manière similaire, nous aimerions tester d'autres types de résolutions et de formulations que celles étudiées ici. En particulier, on peut envisager d'utiliser des solveurs stochastiques (algorithmes génétiques par exemple), des contraintes MTZ alternatives (dynamiques) et des formulations non linéaires (INLP), voire non entières (programmation par contraintes). Il serait aussi intéressant d'intégrer notre algorithme par parcours de graphe pour le tester sur de petites instances, à but de comparaison – section 6.4. Bien entendu, il n'est pas possible de tout tester, mais l'idée est d'envisager d'autres approches alternatives, afin d'interroger certains de nos choix. Le coût d'un tel développement sera supérieur à celui d'un simple changement de solveur, car on touche ici aux formulations elles-mêmes.

Nos travaux de suivi pourraient eux aussi bénéficier de prolongement : outre l'intégration de *Gladys* déjà évoquée, il faudrait « rafraîchir » le code source, y intégrer les améliorations citées au chapitre 9, et l'optimiser un peu – par exemple mieux gérer l'évolution du graphe d'état pour lisser les temps de calcul CPU. Surtout, il faudrait reprendre nos intégrations sur simulateur réaliste (Morse), et pour cela revoir la discrétisation spatiale mise en œuvre pour la remplacer par une discrétisation à base de primitive de mouvement. Nous en attendons des performances plus réalistes, valides sur le plan opérationnel, mais l'impact global est encore flou, notamment sur l'horizon temporel atteignable. Dans tous les cas, cela demande un travail non négligeable, tant théorique que pratique.

Enfin, en guise d'aboutissement de nos travaux algorithimiques et de plusieurs des développements proposés, nous aimerions intégrer conjointement nos algorithmes de patrouille et de suivi dans le simulateur Morse, afin de les faire tourner en parallèle pour des missions de « contrôle de zones » regroupant des objectifs de patrouille (détection) et de suivi de cibles. L'intérêt d'une telle intégration est multiple : c'est une étape supplémentaire dans le processus de validation expérimental qui montrerait *a posteriori* la validité de nos algorithmes. Cela permettrait également de confronter nos algorithmes de patrouille à d'autres métriques en les évaluant directement face à des cibles de niveaux cognitifs différents – voir 2.4 – plutôt qu'à travers l'oisiveté et le non-déterminisme. Une intégration commune permettrait en outre de mettre en œuvre les mécanismes d'allocation de tâches abordés pour le suivi – section 9.5. De plus, une telle intégration menée à bien nécessite d'aborder les aspects de supervision, majoritairement délaissés dans nos travaux. Cette intégration et cette validation conjointes sont donc l'aboutissement espéré de nos contributions algorithmiques, mais elles nécessitent un travail conséquent dans leur mise en œuvre.

Au delà des développements additionnels dans le prolongement direct de nos travaux, les résultats et considérations développés dans ce manuscrit contribuent à la mise en lumière de plusieurs perspectives de recherche à la portée plus étendue que les seuls cadres applicatifs de la patrouille et du suivi. Nous les présentons ci-après, regroupées en deux directions complémentaires.

La première direction porte sur l'exploitation de modèles plus réalistes et plus riches. Nous avons proposé et exploité des modèles multicouches, permettant des considérations nouvelles et des plans plus fins, mais il est possible d'aller encore au delà. Ainsi, la prise en compte de l'incertitude des données peut mener à des nouvelles considérations pour la planification, autorisant de nouveaux objectifs et contraintes, par exemple sur la localisation. À cela, il faut ajouter des modèles de communication plus précis et réalistes. Au regard de notre expérience pratique et des récents développements technologiques, nous ne sommes pas convaincu que les communications doivent s'établir de manière *ad hoc*, deux à deux. En effet, ce genre de réseau, facile à émuler, est techniquement très difficile à mettre en place dans la pratique. En parallèle, les réseaux de communication ne cessent de se développer. Aussi, de nos jours, on peut très bien imaginer que tous les robots soient reliés par 3G ou 4G plutôt que par Wi-Fi ou ondes radio standards. Pour autant, cela ne signifie pas que l'hypothèse de la connectivité parfaite reste réaliste ; simplement, au lieu de chercher des liens de communication entre robots, nous imaginons plutôt des modèles de couverture réseau, avec une qualité minimale à rechercher. De cette manière, la prise en compte des communications aura un impact tangible sur les trajectoires lorsque cela s'avère nécessaire, mais les robots seront indépendants des autres dans leurs capacités de communication.

Par ailleurs, il existe des situations où les communications ne peuvent être établies, soit parce qu'il est nécessaire de rester « silencieux », soit parce que les infrastructures communes ne sont pas disponibles (brouillage, contexte sous-marin, *etc.*). Dans ces cas-là, plutôt que d'établir des communications ponctuelles directes, difficiles à mettre en œuvre, nous pensons qu'il est préférable d'utiliser des méthodes de coopération implicites, basées par exemple sur l'observation ou l'apprentissage, et ne nécessitant pas de communiquer. Ces schémas de coopération, décentralisés, nous semblent réellement porteurs au regard de l'état de l'art et de nos propres travaux. En particulier, nous avons vu que les approches « parallèles » – sans coordination explicite – ne sont pas nécessairement moins adaptées que les approches où la coordination est explicite. En effet, sur des missions à large échelle, le besoin de coordination peut s'avérer très situationnel. On peut alors imaginer une approche hiérarchique, avec une coordination explicite dans les grandes lignes, permettant de résoudre les cas conflictuels *a priori*, et raffinée localement par des processus de coordination implicite. Nous attendons de ce type d'approche un gain de robustesse tout comme une

baisse des contraintes appliquées au robot, menant à de meilleures performances globales en conditions réelles.

La seconde perspective de recherche porte sur les méthodes par échantillonnage et la notion d'instance. Nous désignons par instanciation le fait d'échantillonner un problème de planification pour obtenir un sous-problème, dont la résolution est plus simple et dont les solutions sont applicables au problème de départ. Les méthodes d'échantillonnage ne sont pas nouvelles : la communauté de *motion planning* les exploite intensément pour résoudre des problèmes trop complexes pour les résolutions traditionnelles optimales. Nous pensons que nos travaux, comme d'autres avant nous (voir notre état de l'art), montrent que ces techniques sont généralisables à de nombreux problèmes de planification. L'instanciation est autant un moyen de gérer la complexité que d'apporter de la robustesse à l'approche globale : en effet, le résultat final obtenu étant incertain, sa pertinence et ses performances sont étudiées en moyenne, et sont moins affectées par les imprécisions lors de la réalisation effective des actions planifiées.

Dans ce type d'approche, l'étape d'échantillonnage est cruciale, bien plus que la résolution de l'instance pour laquelle il existe souvent plusieurs alternatives connues et maîtrisées. Le processus d'échantillonnage est donc un problème à part entière, comme en attestent les nombreux travaux sur le sujet en *motion planning*. En particulier, on sait qu'un échantillonnage uniforme et constant est rarement adapté, et qu'il est plus intéressant de s'adapter aux situations rencontrées. Par ailleurs, il nous semble intéressant de s'intéresser aux instances échantillonnées et à leur proximité avec le problème de départ. En effet, lorsque la densité d'échantillonnage tend vers l'infini, l'instance tend vers le problème de départ, et l'on peut espérer que la solution issue de sa résolution optimale converge elle aussi, et ce vers la solution optimale du problème d'origine. Nous entrevoyons ici la possibilité d'obtenir certaines garanties théoriques sur le degré d'optimalité des solutions calculées à travers la distance des instances au problème de départ. Cela reste cependant très prospectif, et sans doute complexe au regard de la difficulté du processus d'échantillonnage et des phénomènes l'affectant.

Troisième partie

ANNEXES

A

LIBRAIRIE GLADYS

The miracle is this : The more we share, the more we have.

— Leonard Nimoy

Gladys est une librairie sous licence BSD développée essentiellement au LAAS et plus récemment à l'Onera par Pierrick Koch (LAAS), Arnaud Degroote (LAAS), Patrick Bechon (Onera) et Cyril Robin (LAAS). *Gladys* est l'acronyme de *Graph Library for Autonomous and DYnamic Systems*. Son développement est le prolongement des idées exprimées au chapitre 5 sur la gestion des modèles et données nécessaires aux processus décisionnel, et notamment à la planification. Actuellement, elle fournit une API de haut niveau pour différents modules de planification, dont un module de calcul d'itinéraire et des modules d'exploration, de patrouille et de suivi*. Cette API permet de s'abstraire des considérations liées aux formats de données pour accéder directement à l'information sémantique dont on a besoin : existence et qualité d'un chemin, d'un lien de visibilité, *etc.* Écrite en Python et C++, *Gladys* s'appuie sur la libraire GDAL pour gérer les modèles d'environnement géoréférencés et sur la librairie Boost Graph pour gérer les structures de graphes :

— `http://github.com/pierriko/gladys`
— `http://gdal.org`
— `http://boost.org/libs/graph`

Nos modules de planification présentés aux chapitres 7, 8 et 9 utilisent une version simplifiée de Gladys pour des raisons historiques (en suivi) ou de contraintes de temps d'intégration (en patrouille). Les APIs utilisées sont néanmoins compatibles, et les données sous-jacentes analogues.

Dans cette annexe, nous présentons quelques uns des formats des fichiers utilisés – section A.1 – pour donner un aperçu de ce que fait *Gladys*, et nous montrons un exemple basique exploitant l'API de la librairie – section A.2. Le lecteur désirant approfondir ou exploiter la librairie est renvoyé à la page Github de *Gladys* ; le code source des modules l'exploitant est également une bonne source d'exemples d'utilisation*.

Ariane (`https://github.com/cyrobin/xares` et Xares (`https://github.com/cyrobin/xares`) sont deux exemples d'exploitation de Gladys.

A.1 DONNÉES D'ENTRÉE

Gladys utilise actuellement deux types de données : des modèles d'environnement sous forme fichiers GDAL géoréférencés – section A.1.1 – et des modèles de robots sous forme de fichiers de description .json – section A.1.2. Nous présentons également ici un exemple de fichier décrivant une mission de patrouille – section A.1.3. Bien que ce type de fichier ne soit pas exploité par *Gladys* mais plutôt par le module de planification exploitant *Gladys*, il permet de montrer comment s'articulent les différents modèles.

A.1.1 *Modèles d'environnement : cartes géolocalisées*

Gladys exploite plusieurs modèles d'environnement relatif aux actions de déplacement et d'observation. Les modèles de communications ne sont actuellement pas encore implémentés. Ces modèles se présentent sous la forme de grilles rectangulaires – structures de *raster* – dans lesquelles chaque case contient une ou plusieurs informations. Ces grilles sont stockées sous formes de fichiers GDAL au format .tiff ou par la combinaison de deux fichiers aux formats .png et .xml, plus pratique à l'usage. Ces fichiers peuvent être multicouches, c'est-à-dire qu'ils peuvent embarquer plusieurs niveaux d'information.

En prenant pour exemple la combinaison d'un fichier .png monocouche et d'un fichier .xml, le premier indique les valeurs de chaque cellule, tandis que le second indique les informations partagées : échelle de la grille, position UTM de l'origine, orientation des axes, sémantique de la couche, *etc.* Cette combinaison peut être virtuellement étendue à tout type de données relatives à l'environnement : types de terrain, hauteur, information de luminosité, qualité du réseau de communication, *etc.* La seule contrainte est la représentation des données en grille de cellules rectangulaires.

Les figures 32 (a) et (b) montrent deux exemples de fichiers .png servant de carte d'accessibilité pour un même environnement. Il est relativement aisé de manipuler les fichiers GDAL à l'aide des binaires fournis. Entre autre, on peut facilement retrouver les informations associées comme illustré avec le fichier d'accessibilité des AGVs pour Caylus – code 1. On y perçoit notamment les fichiers associés (Files), la taille de la grille (Size), le repère utilisé (Coordinate System), la résolution (Pixel Size), les différentes couches (Band), des données personnalisées (Metadata), *etc.*

```
?> gdalinfo caylus.50cm.bw.agv.png
Driver: PNG/Portable Network Graphics
Files: caylus.50cm.bw.agv.png
       caylus.50cm.bw.agv.png.aux.xml
Size is 550, 500
Coordinate System is:
PROJCS["WGS 84 / UTM zone 31N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",3],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AUTHORITY["EPSG","32631"]]
Origin = (398368.000000000058208,4903239.996810000389814)
Pixel Size = (0.500000000000000,-0.500000000000000)
```

```
Metadata:
  AREA_OR_POINT=Area
  CUSTOM_X_ORIGIN=398369.610000
  CUSTOM_Y_ORIGIN=4903128.010000
  TIFFTAG_SOFTWARE=pix4uav
Image Structure Metadata:
  COMPRESSION=JPEG
  INTERLEAVE=PIXEL
  SOURCE_COLOR_SPACE=YCbCr
Corner Coordinates:
Upper Left ( 398368.000, 4903239.997) ( 1d43'35.25"E, 44d16'31.11"N)
Lower Left ( 398368.000, 4902989.997) ( 1d43'35.42"E, 44d16'23.01"N)
Upper Right ( 398643.000, 4903239.997) ( 1d43'47.65"E, 44d16'31.25"N)
Lower Right ( 398643.000, 4902989.997) ( 1d43'47.83"E, 44d16'23.15"N)
Center ( 398505.500, 4903114.997) ( 1d43'41.54"E, 44d16'27.13"N)
Band 1 Block=550x1 Type=Byte, ColorInterp=Red
  NoData Value=-10000
  Image Structure Metadata:
    COMPRESSION=JPEG
Band 2 Block=550x1 Type=Byte, ColorInterp=Green
  NoData Value=-10000
  Image Structure Metadata:
    COMPRESSION=JPEG
Band 3 Block=550x1 Type=Byte, ColorInterp=Blue
  NoData Value=-10000
  Image Structure Metadata:
    COMPRESSION=JPEG
Band 4 Block=550x1 Type=Byte, ColorInterp=Alpha
```

CODE 1 – Exemple de métadonnées d'un fichier GDAL

A.1.2 *Modèles de robot*

Les modèles d'environnement donnés sous forme de grilles sont convolués avec les modèles de robots afin d'obtenir divers graphes – accessibilité, vision – exploités par la planification. *Gladys* est monorobot, au sens où chaque instance de la librairie charge à son initialisation un fichier .json décrivant le robot. Chaque donnée d'environnement est alors traitée à travers le modèle du robot ; cela permet, par exemple, de prendre en compte la taille et la vitesse du robot pour définir les chemins possibles et leurs coûts. Le code 2 montre un exemple de fichier décrivant un AGV.

```json
{
   "platform":{
      "mass":100.0,
      "radius":0.8,
      "velocity":1.0
   },
   "sensor":{
      "name":"log",
      "range":30.0,
      "quality":2.0,
      "fov":6.28,
      "pose":
         {"x":0.1,"y":0.2,"z":0.7,"t":0.0}
   },
   "antena":{
      "comrange":240.0
   }
}
```

CODE 2 – Exemple de modèle de robot (fichier .json).

On distingue dans ce robot trois composantes : *platform* décrit la mobilité du robot (masse, taille et vitesse maximale), *sensor* décrit ses capacités de vision (dont la portée du capteur, sa qualité et sa position par rapport au centre du robot) et *antena* décrit son modèle de communication (actuellement seul la portée est prise en compte). Le format json permet une grande souplesse dans la description du robot, ce qui facilite l'évolution et l'extension des modèles.

A.1.3 *Modèle de mission*

Les modèles de description des missions ne sont pas utilisés par *Gladys*, qui ne connait pas la notion de mission. Nous les mentionnons néanmoins ici pour illustrer comment les modèles d'environnement et d'agents sont pris en compte. Le code 3 décrit une mission de patrouille telle que présentée et testée aux chapitres 7 et 8. En particulier, le fichier spécifie les paramètres de la mission (dont la période $T = 200$) et décrit les protagonistes (*team*) et les modèles associés.

```
{
   "name": "caylus",
   "map": "./maps/caylus.50cm.bw.png",
   "period": 200,
   "team":
      [{
         "name": "mana",
         "description": "./agv.json",
         "accessibility_map": "./maps/caylus.50cm.bw.agv.png",
         "visibility_map": "./maps/caylus.50cm.bw.agv.png",
         "start_pose":{"x":200.5,"y":445.2,"z":0.7,"t":0.0}
         },
        {
         "name": "minnie",
         "description": "./agv.json",
         "accessibility_map": "./maps/caylus.50cm.bw.agv.png",
         "visibility_map": "./maps/caylus.50cm.bw.agv.png",
         "start_pose":{"x":16.1,"y":92.2,"z":0.7,"t":0.0}
         },
        {
         "name": "ressac",
         "description": "./aav.json",
         "accessibility_map": "./maps/caylus.50cm.bw.aav.png",
         "visibility_map": "./maps/caylus.50cm.bw.aav.png",
         "start_pose":{"x":250.5,"y":450.1,"z":50.2,"t":0.0}
         }
   ]
}
```

CODE 3 – Exemple de descriptif de mission (fichier .json).

A.2 API ET EXEMPLES

Pour connaître toute l'API disponible, l'utilisateur est renvoyé au code source et à la documentation de *Gladys*. Le code 4 donne un exemple restreint de l'utilisation de cette API, tiré du module de calcul d'itinéraire *Ariane*.

```cpp
#include <ostream>
#include <exception>
#include <limits> // for numeric_limits::infinity

#include "ariane/ariane.hpp"

/* [...] */

/* ariane::plan ask Gladys for the path from start to goal
 * and then picks up only some points from this (long) path to
 * elaborate an itinary with only some checkpoints. This sparse itinary is
 * more easily handled by the local navigation module than the original path.
 */
gladys::path_t ariane::plan( const gladys::point_xy_t &start,
                      const gladys::point_xy_t &goal ) const {

   /* init */
   gladys::path_t path ;
   gladys::path_t waypoints ;

   const gdalwrap::gdal& map = ng.get_map().get_map() ;

   gladys::point_xy_t utm_start = map.point_custom2utm( start[0], start[1] );
   gladys::point_xy_t utm_goal = map.point_custom2utm( goal[0], goal[1] );

   std::cerr << "[ariane] start is " << gladys::to_string(start)
          << " (utm : " << gladys::to_string(utm_start) << ")"
          << std::endl ;
   std::cerr << "[ariane] goal is " << gladys::to_string(goal)
          << " (utm : " << gladys::to_string(utm_goal) << ")"
          << std::endl ;

   /* compute path */
   std::cerr << "[ariane] Computing path... " << std::endl ;
   try {
      path = ng.astar_search(utm_start, utm_goal) ;
   } catch (std::exception& e) {
      std::cerr << "[ariane] catch exception : " << e.what() << std::endl ;
      std::cerr << "[ariane] Fail to compute a valid path : please check your
           data." << std::endl ;
      return waypoints ;
   }

   /* compute waypoints (= a specific subset of the path) */
   std::cerr << "[ariane] path computed (size = "
          << path.size() << "). Extracting waypoints..." << std::endl ;

   // when no path have been found
   if ( path.empty() )
      return waypoints ;

   unsigned int i = 0 ;
   double cumul_dist = 0, dist;
   gladys::point_xy_t curr = path[i];
   gladys::point_xy_t last = curr;

   i++;
   for ( ; i < path.size() ; i++ ) {
      dist = gladys::distance( last, path[i] ); // euclidian distance
      cumul_dist += gladys::distance( curr, path[i] ); // euclidian distance
      curr = path[i] ;

      if ( cumul_dist > max_step_length || cumul_dist > ( 1 + curb_tolerance ) *
           dist )
      {
         //new waypoint
```

```
65            last = curr ;
66            cumul_dist = 0 ;
67            waypoints.push_back( map.point_utm2custom( last[0], last[1] ));
68         }
69      }
70
71      // always add the last point
72      waypoints.push_back( map.point_utm2custom( curr[0], curr[1] ));
73
74      /* the end */
75      std::cerr << "[ariane] Done." << std::endl ;
76
77      return waypoints ;
78
79   }
```

CODE 4 – Exemple d'utilisation de la librairie *Gladys* par un module de planification (tiré du code source d'*Ariane*).

B

SOLVEURS IP

J'entends, j'oublie.
Je vois, je retiens.
Je fais, je comprends.

— attribué à Confucius

Comme évoqué à la section 6.6.2, il existe de nombreuses façons de résoudre les problèmes d'optimisation en nombres entiers comme ceux – TOP et SP – que nous avons formulés au chapitre 6. Nous avons dans ce chapitre délibérément privilégié les formulations en nombres entiers linéaires (ILP)*. D'autres options s'offraient à nous, dont la programmation par contraintes ou l'optimisation non linéaire. Nous avons privilégié ces formulations car elles sont très étudiées et bien connues, et comparativement aux autres formulations il existe des méthodes efficaces permettant de les résoudre malgré la complexité sous-jacente.

L'optimisation linéaire en nombre entier est NP-difficile. Le cas particulier dans lequel toutes les variables sont binaires est l'un des 21 prolbèmes NP-complet de Karp.

Bien entendu, il serait intéresser de comparer expérimentalement nos formulations MILP avec d'autres types de formulations, car comme nous l'avons vu, la linéarisation des contraintes ($\max$, $\min$ ou les liens logiques par exemple) se fait au prix de variables et contraintes additionnelles, c'est-à-dire au prix d'une certaine complexité. Néanmoins, et comme déjà exprimé, ce qui nous importe est de montrer que notre approche est viable pour l'élaboration de schémas de patrouille en contexte antagoniste. Notre objectif n'est pas d'élaborer de nouvelles méthodes de résolution de ces problèmes, ni de les comparer entre elles.

Une fois le problème exprimé clairement par une formulation MILP, il nous faut ensuite le résoudre. Pour cela, de nombreux solveurs existent, et cette annexe explique nos choix et les considérations associées. Nous présentons et comparons brièvement ici quelques solveurs parmi les plus connus avant d'expliquer notre choix du solveur GLPK – section B.1. Nous tâchons ensuite d'en expliquer le fonctionnement, typique de nombreux solveurs MILP. Nous clôturons cette annexe avec quelques remarques et considérations éclairant certains résultats expérimentaux.

Remarque. Il existe une large littérature et de nombreux travaux expérimentaux concernant l'optimisation linéaire et en nombres entiers (*linear and integer programming*). Nous ne donnons ici que quelques éléments clefs permettant d'éclairer les chapitres concernés, selon la perception de l'auteur.

Il existe de nombreux solveurs MILP efficaces avec des API disponibles dans plusieurs langages. Nous pouvons citer PuLP (Python), Gurobi (Python) ou CPLEX (C/C++) par exemple. Ces solveurs sont généralement des solveurs exacts et déterministes – du moins ils proposent un mode déterministe. À ces méthodes déterministes s'ajoutent des méthodes stochastiques de résolution telles que les algorithmes génétiques ou l'entropie croisée. Ce type de méthode s'est avéré très efficace, mais ne fournit pas de garanties quant à l'optimalité du résultats renvoyés, c'est pourquoi nous ne les avons pas sélectionnées pour résoudre nos problèmes TOP et SP. De plus, comme évoqué précédemment, il est également possible d'élaborer un solveur spécifique aux problèmes formulés, mais nous avons écarté cette option car elle ne cadrait pas avec nos objectifs.

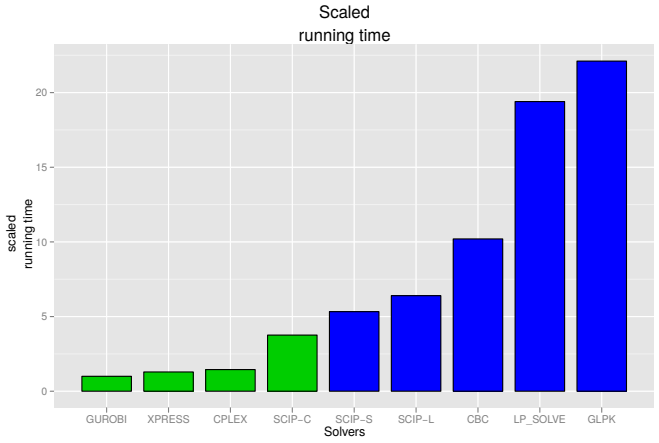
Ainsi nous avons limité notre choix d'une méthode de résolution aux solveurs disponibles, reconnus et déterministes. Parmi ces solveurs, on constate une diversité dans les performances. On peut notamment distinguer deux classes de solveurs : les solveurs commerciaux, disposant d'un large soutien financier pour se développer, et les solveurs libres, aux moyens plus limités, les premiers montrant de meilleures performances.

Il n'existe pas d'échelle typique et universelle permettant de comparer deux solveurs ; la méthode habituellement consiste à tester différents problèmes connus – dont le TSP – en faisant varier le nombre de variables et de contraintes*. En effet, leurs performances sont grandement impactées par ces deux chiffres objectifs permettant d'évaluer dans une certaine mesure la complexité d'un problème. Malheureusement, l'étude de ce nombre n'est pas suffisant, car la « topologie » du problème est tout aussi importante, c'est pourquoi les *benchmarks* disponibles comparent les solveurs sur un ensemble de problèmes classiques différents. La figure 83 montre les performances de plusieurs solveurs reconnus, malheureusement illustrée que pour un seul problème MILP, le *Cell Suppression Problem* (CSP). Nous renvoyons le lecteur désirant approfondir cette comparaison, et plus largement le sujet de l'évaluation des solveurs IP, à l'analyse dont ces graphiques sont tirés [101].

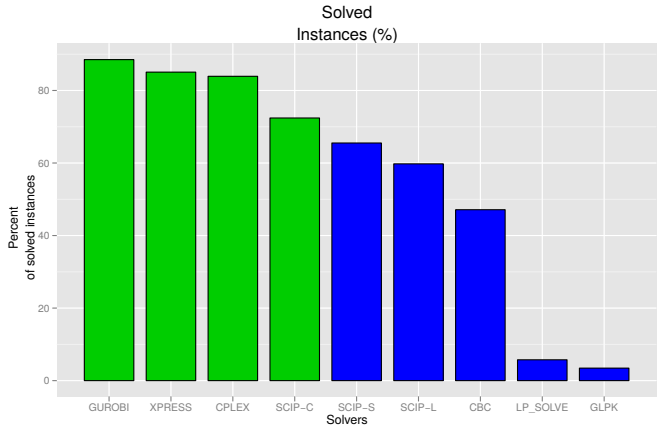
http://miplib.zib.de/miplib2010.php présente un exemple de benchmark *poussé, avec des tests variés.*

Bien que certains de ces solveurs commerciaux proposent des licences gratuites aux laboratoires de recherche, nous avons préféré nous orienter sur les solveurs libres par soucis d'universalité. Plus spécifiquement, nous avons retenu le solveur GLPK (*GNU Linear Programming Kit**), gratuit et *open source*. Il présente également l'avantage d'être compatible avec de nombreux formats de fichiers décrivant les problèmes IP. En particulier, nous trouvons la description des problèmes abordés très « lisible », proche de la formulation mathématique et donc facilement compréhensible et abordable – voir plus de détail à la section suivante. Pour toutes ces raisons, il nous semble bien adapté à la création d'un premier prototype permettant une première étude de la pertinence de nos approches.

GLPK est développé par Andrew Makhorin http://www.gnu.org/software/glpk/glpk.html.



(a) Temps de calcul



(b) Taux de réussite

FIGURE 83 – *Benchmark* des principaux solveurs ILP, prenant en compte (a) leur temps de calcul moyen et (b) leur taux de réussite moyen. Les solveurs commerciaux sont indiqués en vert. Les solveurs libres sont indiqués en bleu. Tiré de [101].

Les capacités du solveur GLPK, ce sont révélées en grande partie suffisantes pour montrer la viabilité de nos formulations TOP. En revanche, ils ont montré leurs limites face aux formulations SP, trop complexes*.

Avec le recul, investir dès le départ dans l'intégration d'un solveur plus efficace aurait peut-être été un choix judicieux.

Remarque. Compte-tenu de l'échec de GLPK face aux formulations SP, mais aussi des résultats prometteurs obtenus en formulation TOP, nous aimerions pour la suite pousser l'intégration de notre prototype en le liant aux solveurs commerciaux CPLEX ou Gurobi, bien plus performants que GLPK. Cela s'accompagnerait d'une réécriture des algorithmes en C++ pour réduire d'avantage les temps de calculs, et d'une intégration complète de la librairie *Gladys*.

B.2 GLPK

AMPL, acronyme de A Mathematical Programming Language, *est un langage de modélisation algébrique permettant de décrire des problèmes mathématiques complexes afin de les résoudre informatiquement.*

GLPK est le solveur linéaire du projet GNU ; il vise à résoudre des problèmes d'optimisation linéaire et en nombres entiers, à large échelle. En tant que partie du projet GNU, GLPK est distribué sous license GNU GPL. Écrit en ANSI C, il prend deux formes : des binaires exécutables, et une librairie C. La description des problèmes y est faite à l'aide d'une sous partie du langage AMPL*. Il existe plusieurs interfaces (*bindings*) vers d'autres langages, dont Python. Nous avons exploité PyMathProg, fournissant une interfaces Python avec une description des problèmes très proche de AMPL. Le code 5 en donne un exemple pour un petit TSP, tiré de la documentation de PyMathProg, accessible à l'adresse http://pymprog.sourceforge.net/.

```
# TSP, Traveling Salesman Problem
# Written in pymprog by Yingjie Lan <ylan@umd.edu>

# The Traveling Salesman Problem (TSP) is a famous problem.
# Let a directed graph G = (V, E) be given, where V = {1, ..., n} is
# a set of nodes, E <subset of> V x V is a set of arcs. Let also each arc
# e = (i,j) be assigned a number c[i,j], which is the length of the
# arc e. The problem is to find a closed path of minimal length going
# through each node of G exactly once.

n = 16 #number of nodes
V = range(1,n+1) #set of notes
#cost or each arc, format: (start, end):cost
c = {(1,2):509, (1,3):501, (1,4):312, (1,5):1019, (1,6):736, (1,7):656,
     (1,8): 60, (1,9):1039, (1,10):726, (1,11):2314, (1,12):479,
    ...
    (16,1):150, (16,2):542, (16,3):499, (16,4):455, (16,5):1033, (16,6):687,
     (16,7):592, (16,8):206, (16,9):933, (16,10):610, (16,11):2248,
     (16,12):417, (16,13):406, (16,14):449, (16,15):636}
#set of arcs: (i,j) repr an arc from i to j
E = c.keys()

import pymprog
p = pymprog.model("tsp")
x = p.var(E, 'x', bool) # x created over index set E.
#minize the total travel distance
p.min(sum(c[t]*x[t] for t in E), 'totaldist')
#subject to: leave each city exactly once
p.st([sum(x[k,j] for j in V if (k,j) in E)==1 for k in V], 'leave')
#subject to: enter each city exactly once
p.st([sum(x[i,k] for i in V if (i,k) in E)==1 for k in V], 'enter')

#We then need some flow constraints to eliminate subtours.
#y: the number of cars carried: endowed with n at city 1.
#exactly one car will be sold at each city.
y=p.var(E, 'y')
p.st([(n-1)*x[t] >= y[t] for t in E], 'cap')
p.st([sum(y[i,k] for i in V if (i,k) in E) + (n if k==1 else 0)
==sum(y[k,j] for j in V if (k,j) in E) + 1 for k in V], 'sale')


p.solve(float) #solve as LP only.
print "simplex done:", p.status()

p.solve(int) #solve the IP problem

# The optimal solution is 6859
print(p.vobj())
```

```
49  tour = [t for t in E if x[t].primal>.5]
50  cat, car = 1, n
51  print("This is the optimal tour with [cars carried]:")
52  for k in V:
53    print cat,
54    for i,j in tour:
55      if i==cat:
56        print "[%g]"%y[i,j].primal,
57        cat=j
58        break
59  print cat
```

CODE 5 – Exemple de formulation du problème de TSP à l'aide de PyMathProg, tiré de la documentation.

B.3 FONCTIONNEMENT

GLPK utilise les méthodes du simplex et des points intérieurs pour résoudre les problèmes non entiers ; il utilise l'algorithme de *Branch and Bound* tout autant que la méthode des plans sécants pour résoudre les problèmes en nombres entiers.

Nous ne décrirons pas ici les méthodes du simplex et des points intérieurs, car il existe une abondante litérature à ce sujet, facilement accessible. Ce qu'il est important de comprendre en revanche, c'est que ces méthodes permettent de résoudre des problèmes d'optimisation en nombres réels – et donc non entiers. Le simplex est utilisé pour les problèmes linéaires, tandis que la méthode des points intérieurs est généralement réservée aux problèmes non-linéaires.

La résolution d'un problème d'optimisation discret, en nombres entiers – *(Mixed) Integer Programming* (MIP) – se passe alors ainsi : on résoud d'abord la relaxation du problème, c'est-à-dire le même problème, mais avec des variables réelles. Cette étape permet d'évaluer plusieurs choses, dont la faisabilité, des redondances dans les contraintes, une structure particulière de l'espace des solutions, *etc.* Puis on cherche à s'appuyer sur les résultats obtenus pour résoudre le problème avec les domaines des variables d'origine à l'aide des méthodes de *branch and bound* et des plans sécants.*

Ce type de résolution en deux temps est tout à fait classique de la plupart des solveurs ILP [96].

Remarque. La relaxation d'un problème change sa complexité en le ramenant à un problème plus général dont on sait évaluer facilement la solution. C'est notamment le cas des problèmes linéaires à variables réelles, que l'on sait résoudre optimalement avec une extrême efficacité.

Branch and Bound

La méthode de *Branch and Bound*, aussi appelée « Séparation et Évaluation » en français, consiste à subdiviser l'espace des variables en sous-espaces – *Branch* ou « séparation » – et à évaluer les sous-problèmes induits – *Bound* ou « évaluation ». Cette évaluation peut prendre deux formes : soit résoudre le sous-problème induit, soit montrer que cet espace ne peut pas comporter la solution optimale au problème sur l'espace global.

Cette méthode est récursive – on peut séparer des sous-espaces en espaces plus petits – et c'est pourquoi la phase d'évaluation est très importante. En particulier, on va essayer de montrer que le sous-espace considéré solutions réalisables ne contient pas de solution optimale. Pour cela, une méthode classique consiste à calculer la borne inférieure – si l'objectif est de minimiser une fonction – à l'aide de la relaxation des contraintes, et notamment du domaine des variables. Dans le cas d'un minimum, si l'on arrive à trouver une borne inférieure présentant un coût supérieur au coût de la meilleure solution trouvée jusqu'à présent, on est alors garanti que le sous-ensemble considéré ne contient pas l'optimum global. Lorsque le sous-espace de solutions donne un problème induit « suffisamment simple », on le résout optimalement.

Méthode des plans sécants

La méthode des plans sécants pour les problèmes d'optimisation linéaires en nombres entiers fonctionne en résolvant d'abord la relaxation linéaire – par exemple avec des variables réelles. La théorie sur l'optimisation linéaire nous indique alors que sous certaines conditions généralement vérifiées, il est toujours possible de trouver une solution optimale dans un point extrême de l'espace de recherche – un « coin ». On teste alors cette solution réalisable pour savoir si elle correspond à une solution en nombres entiers. Si c'est le cas, c'est notre solution au problème. Sinon, nous avons la garantie qu'il existe une inégalité linéaire séparant l'optimum de l'enveloppe convexe du l'ensemble réellement faisable – l'espace des solutions en nombres entiers. Déterminer cette inégalité est un problème de séparation, et une telle inégalité est appelé « plan sécant » (*cut*). Il est alors possible d'ajouter ce plan sécant à la relaxation linéaire du problème ; la solution optimale non entière calculée précédemment n'est alors plus une solution acceptable pour la relaxation. Le processus est alors répété jusqu'à ce qu'une solution entière soit trouvée.

Remarque. GLPK offre le choix entre toutes ces méthodes, certaines pouvant se révéler plus appropriées que d'autres dans certains cas. Nous n'avons pas voulu aborder cet aspect de la résolution qui ne nous semble pas central par rapport à nos objectifs – GPLK ne présente pas de toutes façons les meilleurs performances possibles – et nous avons donc délibérément laissé les paramètres par défaut.

B.4 CARACTÉRISTIQUES

Les aspects « topologiques » d'un problème ILP regroupent notamment la proximité du problème avec sa relaxation, et l'existence de symétrie.

Comme expliqué précédemment, les performances du solveur – taux de réussite et temps de calcul – évoluent selon le nombre de variables et de contraintes, mais également selon le type de problèmes ILP à résoudre – selon la « topologie* » du problème. Cet aspect topologique étant intrinsèque à chaque problème, nous avons surtout fait varier dans nos tests

expérimentaux le nombre de variables et contraintes à l'aide des différents paramètres – voir la section 7.2.1 – et considéré des environnements de tests variés.

Par ailleurs, il est important de garder à l'esprit certaines caractéristiques du solveur GLPK. Il est notamment déterministe* dans ces paramètres par défaut. De plus, et grâce à la résolution en deux temps, le solveur est capable d'estimer très vite, à travers la relaxation, si des solutions existent ou non. Il faut également noter qu'il est correct, sous réserve qu'on lui accorde un temps de calcul suffisant (pouvant être *très* élevé), c'est-à-dire qu'il est renvoi une solution optimale s'il en existe, pour peu qu'on lui laisse le temps d'en trouver. En pratique, nous avons constamment utilisé une limite de temps : le solveur ne fournit alors alors aucune garantie de renvoyer une solution correcte, même sous-optimale.

Notre approche est stochastique car les problèmes IP sont créés – instanciés – stochastiquement ; la résolution est, elle, déterministe.

Enfin, il faut noter que les problèmes abordés ici sont NP-complets. Si le temps de calcul nécessaire aux solveurs n'est pas exponentiel comme le serait un algorithme naïf d'itération des solutions, les solveurs ont malgré tout un temps de réponse non-linéaire avec le nombre de variables et de contraintes du problème. En pratique, les performances semblent polynomiales, bien que nous n'ayons pas estimé cela précisément. Ce ressenti est particulièrement fort avec la qualité des solutions renvoyées (optimales, sous-optimales ou pas de solution trouvée*).

GLPK qualifie la qualité des solutions renvoyées : undef *pour pas de solutions,* feas *pour une solution sous-optimale, et* opt *pour une solution optimale.*

Cette évolution du temps de calcul face au nombre de variables et de contraintes est très importante car elle met en perspective toute augmentation ou diminution de ce nombre ; en particulier, le fait de diviser la complexité par N, le nombre de robots, dans le cas d'une résolution décentralisée – voir chapitre 8 – nous apporte un gain de performances important, avec des instances résolues beaucoup plus rapidement et de manière optimale.

C

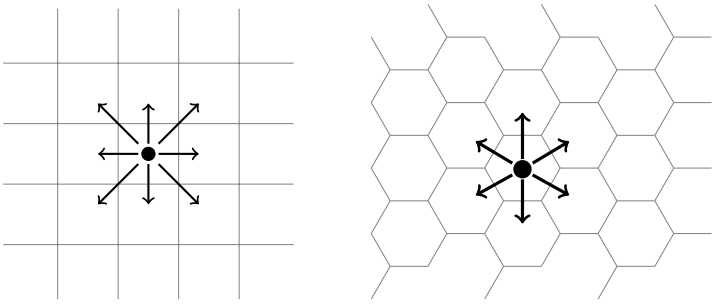
COMPLEXITÉ DU SUIVI

Convincing yourself is easy,
persuading a colleague is harder,
but proving it to a computer is hardest of all !

— attribué à R. W. Hamming

Cette annexe se rapporte au chapitre 9 et développe les résultats de complexité qui y sont avancés, notamment dans la section 9.3. La preuve qui suit est vraie pour un environnement sans obstacle, sur une grille en 9-connexité*, par exemple une grille cartésienne carrée – figure 84a – prenant en compte les diagonales. De manière contre-intuitive peut-être, cette complexité est donc une borne supérieure, notamment parce que les obstacles, les restrictions sur la perception ou une connexité plus réduite sont autant de restrictions aux mouvements et donc aux nombres de configurations spatiales possibles.

La 9-connexité correspond à une grille carrée prenant en compte les diagonales entre cases et le mouvement « rester sur place » (figure 84a).



(a) 9-connexité en grille carrée (b) 7-connexité en grille hexagonale

FIGURE 84 – Grilles et connexités : (a) une grille carrée en 9-connexité (8 cases adjacentes + elle-même) ; (b) une grille hexagonale en 7-connexité (6 cases adjacentes + elle-même).

Démonstration. Sous de telles hypothèses, en reprenant toutes les considérations de redondances spatiales mises en avant dans la section 9.3, nous avons à chaque étape* θ de construction du graphe de poursuite obtenu au plus n_θ nœuds avec $n_\theta \leqslant |Acc(a_{cible}, \theta)| * |Acc(r_p, \theta)|$ où $|Acc(a, \theta)| = A_{a,\theta}$ est l'ensemble des positions accessibles pour a au temps θ. La taille total n_{tot} du graphe est donc inférieure à $\sum_{\theta=1}^{T} A_{a_{cible},\theta} * A_{r_p,\theta}$ où T et l'horizon temporel considéré. Par ailleurs, en 9-connexité, nous avons $A_{a,\theta} \leqslant 1 + \sum_{\sigma=1}^{\theta} 8\sigma = (2\theta+1)^2$ (l'égalité est atteinte s'il n'y a pas d'obstacle), d'où $n_{tot} \leqslant \sum_{\theta=1}^{T}(2\theta+1)^4 = O(T^5)$. Nous considérons ici que le poursuivant et la cible ont la même vélocité : cela ne restreint

Nous appelons étape *l'augmentation de l'horizon temporel d'un pas de temps. Cela correspond aussi à une étage additionnel de l'arbre d'état.*

pas la généralité de la preuve mais la simplifie, car différentes vélocités ne changent que la complexité résultante $|A_{a_{cible},\theta}| \neq |A_{r_p,\theta}|$ et non le raisonnement y conduisant.

Avec l'ajout des boucles temporelles et de la construction itérative du graphe de poursuite, il suffit de développer les nouveaux nœuds à chaque étape, c'est-à-dire les nœuds feuilles. En reprenant les mêmes hypothèses, nous avons un nombre de nœuds feuilles n_f vérifiant :

$$\begin{aligned} n_f &\leqslant \left(A_{a_{cible},T} - A_{a_{cible},T-1}\right) * A_{r_p,T-1} \\ &\quad + \left(A_{r_p,T} - A_{r_p,T-1}\right) * A_{a_{cible},T-1} \\ &\quad + \left(A_{a_{cible},T} - A_{a_{cible},T-1}\right) * \left(A_{r_p,T} - A_{r_p,T-1}\right) \\ &\leqslant 8T * O(T^2) + 8T * O(T^2) + 8T * 8T \\ n_f &\leqslant O(T^3) \end{aligned}$$

□

La preuve peut se généraliser à plusieurs types de grilles, avec des complexités variant légèrement. Ainsi, dans le cas d'une grille hexagonal en 7-connexité (figure 84b), on a une complexité de $n_{tot} = O\left(T^5\right)$ et $n_f = O\left(T^3\right)$.

Démonstration. On reprenant le cheminement de la preuve précédente, on a cette fois-ci : $A_{a,\theta} \leqslant 1 + \sum_{\sigma=1}^{\theta} 6\sigma = 3\theta^2 + 3\theta + 1$, d'où $n_{tot} \leqslant \sum_{\theta=1}^{T}(3\theta^2 + 3\theta + 1)^2 = O(T^5)$ également, avec des coefficients cependant plus avantageux que pour une grille carrée, notamment sur les puissances intermédiaires.

De même, le nombre de nœuds-feuilles vérifie maintenant :

$$\begin{aligned} n_f &\leqslant 6T * O(T^2) + 6T * O(T^2) + 6T * 6T \\ n_f &\leqslant O(T^3) \end{aligned}$$

□

$\alpha^r_{p_1p_2}$ Qualité du signal de communication avec p_2 depuis p_1 pour le robot r, page 89

$\bar{y}_j$ Variables indiquant si p_j n'est pas visité par le plan courant ($\bar{y}_j = (1 - y_j)$, page 113

β^r_i Variables binaires additionnelles utilisées dans les SEC MTZ, page 109

η^r Fonction de pseudo densité de probabilité basée sur l'utilité des positions $q \in Q$, page 98

ν^r Fonction d'utilité cumulée basée sur u, P^r et ϕ^r, page 98

ϕ^r_{pq} Qualité de perception de q depuis p par le robot r, page 89

$\pi^a(p)$ Probabilité de présence de l'agent a en p, page 89

$\{X\}$ Espace de recherche des solutions en formulation IP (= ensemble des plans potentiels X), page 104

A L'ensemble des agents protagonistes de la mission en cours, page 88

a Un agent protagoniste de la mission en cours, page 88

$a_s(r,t)$ Fonction d'activation : indique si le robot $r \in R$ activement sollicité au temps $t \in \mathcal{T}$ pour la tâche s considérée, page 178

a^r_{ij} Variable binaire indiquant si l'arc a^r_{ij} de $\mathcal{G}^r$ est traversable par le robot r, page 96

b^r Facteur de branchement du graphe $\mathcal{G}^r$, page 103

$b_{jk}r$ Variables binaires utilisées dans la linéarisation de certaines formulations IP, page 108

$c^r_{p_1p_2}$ Coût de déplacement de p_1 à p_2 pour le robot r, page 88

d^r_i Spécifie la position des nœuds dans le parcours (départ, intermédiares,arrivée), page 106

d_{cr} Distance indiquant la portée maximale de communication (*communication range*), page 172

$dist(p_1, p_2)$ Distance (généralement eulcidienne) entre deux positions p_1 et p_2, page 121

F Fonction objectif en formulation IP, page 104

f Fonction arbitraire transformant la fonction d'utilité en fonction de pseudo densité de probabilité η^r, page 98

G Contraintes en formulation IP, page 104

$g^u(q)$ Le taux de croissance de l'utilité u en $q \in Q$; par extension, indique l'importance de la position q, page 121

H Contraintes en formulation IP, page 104

i Indice des positions accessibles $p_i \in P$; par extension, identifie le sommet de $\mathcal{V}$ associé à la position p_i, page 96

j Indice des positions accessibles $p_j \in P$; par extension, identifie le sommet de $\mathcal{V}$ associé à la position p_j, page 96

k Indice des positions observables $q_k \in Q$, page 95

M Grande constante arbitraire utilisée dans la linéarisation de certaines formulations IP, page 108

m Le nombre de positions observables ($m = card(Q)$), page 95

N Le nombre de robots disponible pour la mission ($N = card(R)$), page 95

n^r Cardinal de P^r lorsqu'il est fini (discret) ; par extension, désigne aussi le nombre de sommets de $\mathcal{V}^r$, page 96

p Une position accessible, page 88

P^a L'ensemble des positions accessibles à l'agent a, page 88

P^r L'ensemble des positions accessibles au robot r, page 88

Q L'ensemble des positions observables ; généralement lié à l'objectif de la mission, page 88

q Une position observable, page 88

R L'ensemble des robots disponibles pour la mission en cours, page 88

r Un robot disponible pour la mission en cours, page 88

R_θ^{act} En suivi, désigne l'ensemble des robots actifs (poursuivant et support) au temps $\theta \in \mathcal{T}$, page 178

r_p En suivi, désigne le robot poursuivant la cible a_{cible}, page 180

T Durée maximale d'une phase de patrouille (= coût maximal d'un plan) ; horizon temporel de la planification, page 96

u Utilité (la définition précise varie selon le contexte), page 90

X Un plan potentiel en formulation IP ; l'ensemble des X forme l'espace de recherche des solutions, page 104

x_{ij}^r Variable binaire indiquant si l'arc a_{ij}^r est parcouru par le robot r, page 104

y_j Indique si la position p_j est visité par l'équipe de robots, page 106

y_k Désigne la meilleure observation de q_k par l'équipe de robots en charge de la mission, page 108

$\mathcal{A}$ Arcs du graphe $\mathcal{G}$, page 96

$\mathcal{A}^{r}$ Arcs du graphe d'accessibilité $\mathcal{G}^{r}$, page 96

$\mathcal{G}$ Graphe formé par le produit des graphes $\mathcal{G}^{r}$, page 96

$\mathcal{G}^{r}$ Graphe d'accessibilité de r, constitué par les $p_i^r \in P^r$, page 96

$\mathcal{P}^{r}$ L'ensemble des positions $p \in P^r$ formant le plan élaboré pour le robot r dans le cadre de sa mission, page 121

$\mathcal{T}$ Désigne l'espace temporel considéré $\subset \mathbb{R}^+$ (principalement utilisé en suivi), page 178

$\mathcal{V}$ Sommets du graphe $\mathcal{G}$, page 96

$\mathcal{V}^{r}$ Sommets du graphe d'accessibilité $\mathcal{G}^{r}$, page 96

GLOSSAIRE

Anytime — Un algorithme *anytime* est capable de donner une solution valide à un problème même s'il est interrompu avant d'avoir terminé. L'algorithme trouve de meilleures solutions au fur et à mesure de son exécution. Sa flexibilité en temps et en ressources sont des propriété très appréciées en robotique mobile. 19, 48, 69

GDAL — La bibliothèque GDAL (*Geospatial Data Abstraction Library*) est une bibliothèque libre permettant de lire et de traiter un très grand nombre de format d'images géographiques (notamment GeoTIFF) depuis des langages de programmation tels que C, C++ et Python. Cette bibliothèque est un des piliers des systèmes d'informations géographique libres, car elle permet d'assurer la compatibilité avec de nombreux systèmes commerciaux reposant sur des formats propriétaires tout autant que sur les normes de l'Open Geospatial Consortium. 87, 119, 120, 205

AAV — Véhicule aérien autonome (*Autonomous Aerial Vehicle*). 1, 21, 25, 28, 30, 43–45, 49, 53, 61, 64, 84, 85, 121, 130, 179, 182, 185, 192

AGV — Véhicule terrestre autonome (*Autonomous Ground Vehicle*). 1, 21, 43, 44, 49, 61, 64, 84, 85, 121, 130, 142, 165, 182, 192

API — En informatique, une interface de programmation ou *API* (pour *Application Programming Interface*) est un ensemble normalisé de classes, de méthodes ou de fonctions servant de façade par laquelle un programme offre des services à d'autres programmes. 83–86, 112, 122, 205, 212

AUV — Véhicule sous-marin autonome (*Autonomous Underwater Vehicle*). 65

Capture
: Au sens de notre taxonomie, le problème de capture (*capture*) a pour but est de « nettoyer » une zone connue et délimitée tout en fournissant des garanties au pire cas, ce qui signifie que toute cible se trouvant dans la zone définie sera trouvée quelles que soient ses capacités. L'optimalité et la complétude sont des caractéristiques essentielles de ce type de problème. 7–9, 17, 18, 20–24, 26, 32, 39, 40, 56, 57, 60

Chasse
: Au sens de notre taxonomie, les problèmes de chasse (*hunting*) désignent les problèmes pour lesquels aucune garantie n'est donnée quant à la détection ou la capture des cibles. Cette absence de garantie vient d'un manque de ressources (robots, temps) mais aussi d'informations, en l'absence desquelles on ne peut fournir ni exploiter de modèles probabilistes sur la position des cibles. 7–9, 39, 41, 45, 58

Couverture de zones
: Au sens de notre taxonomie, l'objectif d'une couverture de zones (*coverage*) est de déterminer les positions optimales d'un ensemble de capteurs fixes afin de surveiller une zone définie. La forme traditionnelle de la couverture de zones est le célèbre problème des gardiens de musée (*the Art Gallery problem*). 7, 8, 13–16, 31, 32, 35, 53

Couverture par ensemble
: Le problème de couverture par ensemble, ou *Set Cover Problem (SCP)* en anglais, est une question classique en analyse combinatoire : étant donné un ensemble d'élément appelé univers et un ensemble S de n sous-ensemble dont l'union est égale à l'univers, le problème de couverture par ensemble consiste à identifier le plus petit sous-ensemble de S dont l'union est égale à l'univers. 14, 16

Déménageur de piano
: Mathématiquement, le problème du déménageur de piano est défini ainsi : étant donnés un ouvert U dans un espace à n dimensions et deux compacts C_0 et C_1 de U, avec C_1 dérivé continuement de C_0, est-il possible de se déplacer de C_0 à C_1 tout en restant entièrement à l'intérieur de U ? [158]. 50

Espace CAT(k)
: En mathématique, un espace CAT(k), k réel, est une classe spécifique d'espace métrique. Les espaces CAT(0) sont aussi connus sous le nom d'espace d'Hadamard, et les espaces euclidiens traditionnels sont un sous-ensemble des espaces d'Hadamard.. 49, 62

Force brute
: La recherche par force brute (*brute-force*) est une méthode de recherche exhaustive constitant à énumérer systématiquement toutes les solutions candidates possibles en vérifiant si elles satisfont ou non les contraintes du problème. Ce type de recherche de solution est facile à implémenter et trouvera toujours une solution si elle existe, mais son coût est proportionnel au nombre de solutions candidates, ce qui peut vite la rendre inutilisable en pratique. 67

Fouille
: Au sens de notre taxonomie, les problèmes de fouille (*probabilistic search*) ont pour but de « nettoyer » au mieux une zone connue comme en capture, mais sans pouvoir fournir de garanties au pire cas. Ces probèmes sont caractérisés par des considérations probabilistes et fournissent des garanties en accord. Ce type de considérations est principalement motivé par un manque de ressources (manque de robots ou de temps) qui ne permet pas de gérer les pires cas. 7–9, 23–32, 39, 40, 57, 63, 116

Gardiens de musée
: Le problème des gardiens de musée (*art gallery problem*) est un problème de visibilité classique en géométrie calculatoire. Son origine renvoie au problème réel du gardiennage des musées dans lequel on cherche le nombre minimal de gardiens nécessaires pour surveiller simultanément la totalité du musée. La géométrie du musée est généralement représenté par un polygone, les gardiens étant chacun un point de ce polygone. 13–16, 18, 20, 35, 226

IP
: *Integer Programming*. 2, 104, 111, 115, 118, 119, 122, 161

LIDAR — La télédétection par laser ou LIDAR (*LIght Detection And Ranging*), est une technologie de télédétection ou de mesure optique basée sur l'analyse des propriétés d'un faisceau renvoyé vers son émetteur. C'est une sorte de « radar laser ». 20, 65, 79

Localisation de cibles — Au sens de notre taxonomie, les problèmes de localisation de cibles (*target localisation*) consistent à traquer une cible à l'aide de plusieurs capteurs afin d'augmenter les connaissances sur la cible en question, notamment la précision de sa position estimée. Ce sont donc fondamentalement des problèmes multirobots. 7, 10, 23, 43–46, 53, 57

LUT — Une *Looked-Up Table* (LUT), aussi appelée table de correspondance, est une liste d'association de valeur. Dans notre cas, elle nous sert principalement à faire correspondre des couleurs avec des valeurs d'utilité. 129, 131

MDP — Un processus de décision markovien (ou MDP, de l'anglais *Markovian Decision Process*), est un modèle de décision stochastique : il peut être vu comme une chaîne de Markov à laquelle on ajoute une composante décisionnelle.. 58, 229

MILP — *Mixed Integer Linear Programming*. 104, 112

MINLP — *Mixed Integer Non-Linear Programming*. 46

MTZ — Les SEC MTZ forment un jeu de contraintes additionnelles permettant d'éliminer les sous-tours dans les solutions des problèmes de types TSP ; elles ont été proposées par Miller, Tucker et Zemlin [104]. 109, 110

Observation
: Au sens de notre taxonomie, les problèmes d'observation (*observation*) sont définis ainsi : étant donnés une équipe de plusieurs robots et plusieurs cibles mobiles désignées, comment déplacer les robots pour observer simultanément l'ensemble des cibles, ou à défaut minimiser le temps pendant lequel une cible n'est pas observée par au moins un robot ?. 7, 10, 15, 45, 51, 53, 57

Patrouille
: Au sens de notre taxonomie, lorsque la recherche active de cible (avec capteurs mobiles) se répète dans le temps, on emploie le terme de problèmes de patrouille (*patrolling*). La patrouille est comme une version cyclique de la fouille et implique une analyse statistique des performances au cours du temps. Elle considère notamment le temps écoulé entre deux observations ou « visites » d'une même position. 7–9, 32–38, 60, 70, 92, 93

PDDL
: PDDL, acronyme anglais de *Planning Domain Definition Language*, est un langage conçu dans le but de standardiser l'expression des problèmes « classiques » de planification en Intelligence Artificielle. Son développement remonte à la fin des années 90. Il existe, un peu ironniquement pour un standard, de nombreux successeurs, variantes et extensions. Voir aussi Wikipedia pour une introduction plus complète. 59, 87

PDF
: *Probability Density Function*. 97–99, 122

POMDP
: Un processus de décision markovien partiellement observable, ou POMDP (de l'anglais *Partially Observable Markovian Decision Process*), est un modèle de décision stochastique dérivé des processus de décision markoviens (MDP) mais pour lesquels l'incertitude est double : l'effet des actions que l'on entreprend est incertain, mais de plus on ne dispose que d'indices pour connaître l'état dans lequel on se trouve, et donc pour décider (l'état est « partiellement observable »). 24, 25, 27, 58

SEC
: *Subtours Elimination Constraintes*. 109

SLAM — *Simultaneous Localization And Mapping*. 68, 83, 86

SP — Le *Sightseeing Problem* (SP) tire son nom des activités touristiques dans lesquelles l'objectif est de profiter au maximum d'un ensemble de sites intéressants dans un temps donné. Pour cela, il faut faire une sélection des sites observés et de leurs points d'observation. 123–125, 127, 128, 138, 161

Suivi — Au sens de notre taxonomie, les problèmes de suivis (*following*) désignent les problèmes de pistages monorobots/monocibles. Les problèmes de suivi sont très couramment appelés « problèmes de poursuite-évasion », et la forme traditionnelle du problème de poursuite en mathématique est connue sous le terme de « problèmes de l'homme et du lion » (*Lion and Man*). 7, 10, 20, 21, 39, 40, 46, 47, 50, 51, 53, 59, 66, 70, 92, 176

TOP — L'*Orienteering Problem* (OP) tire son nom des courses d'orientation. Dans ce sport, les compétiteurs démarrent à un point de control spécifié et essaient de passer par autant de balises (*checkpoints*) que possible avant de retourner au point de contrôle, le tout dans une fenêtre de temps donnée. Chaque balise rapporte un certain nombre de points, et le but est de maximiser le score total. En équipe, on parle de *Team Orienteering Problem* (TOP). 105, 108, 112, 123–128, 134, 138, 161, 164

TSP — Le problème du voyageur de commerce ou TSP (*Traveling Salesman Problem*) est un problème mathématique qui consiste, étant donné un ensemble de villes séparées par des distances données, à trouver le plus court chemin qui relie toutes les villes. Il s'agit d'un problème d'optimisation pour lequel on ne connait pas d'algorithme permettant de trouver une solution exacte en un temps polynomial. De plus, la version décisionnelle de l'énoncé (pour une distance D, existe-t-il un chemin plus court que D passant par toutes les villes ?) est connue comme étant un problème NP-complet. 32, 33, 41, 48, 60, 83, 84, 104, 105, 108–110, 116, 123

Velodyne	En robotique, la société Velodyne Inc. fut l'une des premières entreprises à proposer un LIDAR rotatif (à 360°). Par métonymie, *Velodyne* désigne couramment les capteurs commercialisés sous cette marque. 79
WiSAR	*Wilderness Search And Rescue*. 61

BIBLIOGRAPHIE

[1] Jose J. Acevedo, Begona C. Arrue, Ivan Maza, and Anibal Ollero. A decentralized algorithm for area surveillance missions using a team of aerial robots with different sensing capabilities. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4735–4740, 2014.

[2] Noa Agmon, Sarit Kraus, and Gal a. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2339–2345, May 2008. ISBN 978-1-4244-1646-2. doi : 10.1109/ROBOT.2008.4543563.

[3] Mohamed Al Marzouqi and Ray a. Jarvis. Robotic covert path planning : A survey. *2011 IEEE 5th International Conference on Robotics, Automation and Mechatronics (RAM)*, pages 77–82, September 2011. doi : 10.1109/RAMECH.2011.6070460.

[4] S. Alamdari, E. Fata, and S. L. Smith. Persistent monitoring in discrete environments : Minimizing the maximum weighted latency between observations. *International Journal of Robotics Research (IJRR)*, October 2013. ISSN 0278-3649. doi : 10.1177/0278364913504011.

[5] Rachid Alami, Frédéric Robert, Félix Ingrand, and Sho'ji Suzuki. Multi-robot cooperation through incremental plan-merging. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2573–2579, 1995. ISBN 0-7803-1965-6. doi : 10.1109/ROBOT.1995.525645.

[6] S Alexander, R Bishop, and R Ghrist. Capture pursuit games on unbounded domains. *L'Enseignement Mathématique 55*, pages 103–125, 2009.

[7] S Alexander, R Bishop, and R Ghrist. Total curvature and simple pursuit on domains of curvature bounded above. *Geometriae Dedicata*, pages 1–15, 2010.

[8] Stephanie Alexander, Richard Bishop, and Robert Ghrist. Pursuit and evasion in non-convex domains of arbitrary dimensions. In *Proceedings of Robotics : Science and Systems (RSS)*, 2006.

[9] Alessandro Almeida, Geber Ramalho, Hugo Santana, Vincent Corruble, and Yann Chevaleyre. Recent Advances on Multi-Agent Patrolling. *Advances in Artificial Intelligence*, 2004.

[10] Brian Alspach. Searching and sweeping graphs : a brief survey. *Le matematiche*, LIX : 5–37, 2006.

[11] Francesco Amigoni and Nicola Basilico. A Decision-Theoretic Framework to Select Effective Observation Locations in Robotic Search and Rescue Scenarios. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[12] Francesco Amigoni, Nicola Basilico, and Nicola Gatti. Finding the Optimal Strategies for Robotic Patrolling with Adversaries in Topologically-Represented Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[13] Francesco Amigoni, Nicola Basilico, and AQ Li. How much worth is coordination of mobile robots for exploration in search and rescue ? *RoboCup 2012 : Robot Soccer World Cup XVI*, 2013.

[14] Jonathan Annas and Jing Xiao. Intelligent Pursuit & Evasion in an Unknown Environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4899–4906, 2009. ISBN 9781424438044.

[15] Jonathan Annas and Jing Xiao. Intelligent Pursuit & Evasion in Unknown Environments Against Human Players - extended abstract. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[16] Hannaneh Najd Ataei, Koorush Ziarati, and Mohammad Eghtesad. A BSO-Based Algorithm for Multi-robot and Multi-target Search. *Recent Trends in Applied Artificial Intelligence*, pages 312–321, 2013.

[17] S Balakirsky. Usarsim : a robocup virtual urban search and rescue competition. *Defense and Security Symposium*, 2007.

[18] T. Bandyopadhyay and M.H. Ang. A greedy strategy for tracking a locally predictable target among obstacles. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2342–2347, 2006. doi : 10.1109/ROBOT.2006.1642052.

[19] T. Bandyopadhyay, Yuanping Li, M.H. Ang, and D. Hsu. A greedy strategy for tracking a locally predictable target among obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2342–2347, May 2006.

[20] Tirthankar Bandyopadhyay, Marcelo H Ang Jr, and David Hsu. Stealth Tracking of an Unpredictable Target among Obstacles. *Algorithmic Foundations of Robotics VI*, pages 43–58, 2005.

[21] Tirthankar Bandyopadhyay, Nan Rong, Marcelo Ang, David Hsu, and Wee Sun Lee. Motion Planning for People Tracking in Uncertain and Dynamic Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[22] Tirthankar Bandyopadhyay, MH Ang Jr, and David Hsu. Motion planning for 3D target tracking among obstacles. *Robotics Research*, 2011.

[23] Stanley J. Benkoski, Michael G. Monticino, and James R. Weisinger. A survey of the search theory literature. *Naval Research Logistics (NRL)*, 38(4) :469–494, 1991.

[24] S. Bhattacharya and S. Hutchinson. On the Existence of Nash Equilibrium for a Two-player Pursuit–Evasion Game with Visibility Constraints. *International Journal of Robotics Research (IJRR)*, 29(7) :831–839, December 2009. ISSN 0278-3649. doi : 10.1177/0278364909354628.

[25] S. Bhattacharya and S. Hutchinson. On the existence of nash equilibrium for a two player pursuit-evasion game with visibility constraints. In *Algorithmic Foundation of Robotics VIII*, volume 57 of *Springer Tracts in Advanced Robotics*, pages 251–265. Springer, 2009.

[26] S. Bhattacharya, S. Candido, and S. Hutchinson. Motion strategies for surveillance. In *Proceedings of Robotics : Science and Systems (RSS)*, 2007.

[27] S. Bhattacharya, R. Ghrist, and V. Kumar. Multi-robot coverage and exploration on Riemannian manifolds with boundaries. *International Journal of Robotics Research (IJRR)*, 33(1) :113–137, October 2013. ISSN 0278-3649. doi : 10.1177/0278364913507324.

[28] Sourabh Bhattacharya, Salvator Candido, and Seth Hutchinson. Motion Strategies for Surveillance. In *Proceedings of Robotics : Science and Systems (RSS)*, 2007.

[29] Lélia Blin, Janna Burman, and Nicolas Nisse. Nettoyage perpétuel de réseaux. In *14èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)*, 2011.

[30] Shaunak D. Bopardikar, Francesco Bullo, and Joao P. Hespanha. Cooperative pursuit with sensing limitations. In *American Control Conference (ACC)*, pages 5394–5399, July 2007. ISBN 1-4244-0988-8. doi : 10.1109/ACC.2007.4282474.

[31] Richard Borie, Craig Tovey, and Sven Koenig. Algorithms and complexity results for graph-based pursuit evasion. *Autonomous Robots*, 31(4) :317–332, September 2011. ISSN 0929-5593. doi : 10.1007/s10514-011-9255-y.

[32] Redouane Boumghar. *Strategies d'acquisition d'information pour la navigation autonome cooperative en environnement inconnu*. PhD thesis, University of Toulouse, 2013.

[33] Rodney A Brooks. How to build complete creatures rather than isolated cognitive simulators. *Architectures for intelligence*, pages 225–239, 1991.

[34] Peter Brugger. Variables that influence the generation of random sequences : An update. *Perceptual and motor skills*, 84(2) :627–661, 1997.

[35] ZS Cai, LN Sun, HB Gao, and PC Zhou. Multi-robot Cooperative Pursuit Based on Task Bundle Auctions. *Intelligent Robotics and Applications*, pages 235–244, 2008.

[36] Zhiqiang Cao, Min Tan, Lei Li, Nong Gu, and Shuo Wang. Cooperative Hunting by Distributed Mobile Robots Based on Local Interaction. *IEEE Transactions on Robotics*, 22 (2) :403–407, 2006.

[37] DW Casbeer and DB Kingston. Cooperative forest fire surveillance using a team of small unmanned air vehicles. *International Journal of Systems Science*, 37(6), 2006.

[38] I Chao, Bruce L Golden, Edward A Wasil, et al. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3) :475–489, 1996.

[39] Han-lim HL Choi, Luc Brunet, and Jonathan P How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4) :912–926, 2009.

[40] Hoang-Nam Chu, Arnaud Glad, Olivier Simonin, Francois Sempe, Alexis Drogoul, and François Charpillet. Swarm approaches for the patrolling problem, information propagation vs. pheromone evaporation. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 442–449, 2007.

[41] Timothy H. Chung, Geoffrey a. Hollinger, and Volkan Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4) :299–316, July 2011. ISSN 0929-5593. doi : 10.1007/s10514-011-9241-4.

[42] Justin Clark and Rafael Fierro. Mobile robotic sensors for perimeter detection and tracking. *ISA transactions*, 46(1) :3–13, February 2007. ISSN 0019-0578. doi : 10.1016/j.isatra.2006.08.001.

[43] David T. Cole, Paul Thompson, Ali Haydar Göktogan, and Salah Sukkarieh. System development and demonstration of a cooperative UAV team for mapping and tracking. *International Journal of Robotics Research (IJRR)*, 29(11) :1371–1399, 2010. doi : 10.1177/0278364910364685.

[44] Anthony Cowley, Hwa-Chow Hsu, and Camillo J. Taylor. Distributed sensor databases for multi-robot teams. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[45] K. Daniel, R. Borie, S. Koenig, and C. Tovey. ESP : Pursuit Evasion on Series-Parallel Graphs (Extended Abstract). In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 4–5, 2010.

[46] Jason Derenick, John Spletzer, and Ani Hsieh. An Optimal Approach to Collaborative Target Tracking with Performance Guarantees. *Journal of Intelligent and Robotic Systems*, 56(1-2) :47–67, January 2009. ISSN 0921-0296. doi : 10.1007/s10846-008-9302-x.

[47] M Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination : A survey and analysis. *Proceedings of the IEEE*, 94(7) :1257–1270, 2006.

[48] Christian Dornhege, Alexander Kleiner, and Andreas Kolling. Coverage Search in 3D. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–8, October 2013. ISBN 978-1-4799-0880-6. doi : 10.1109/SSRR.2013.6719340.

[49] Joseph W Durham, Antonio Franchi, and Francesco Bullo. Distributed Pursuit-Evasion without Mapping or Global Localization via Local Frontiers. *Autonomous Robots*, 2011.

[50] J. H. Eaton and LA Zadeh. Optimal pursuit strategies in discrete-state probabilistic systems. *Journal of Fluids Engineering*, 84(1) :23–29, 1962.

[51] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating Complex Robotic Scenarios with MORSE. In *Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 197–208. Springer, 2012.

[52] F Ehlers. Multi-Robot Teamwork in Multistatic Sonar. In *ICRA Workshop : Search and Pursuit/Evasion in the Physical World*, 2010.

[53] Thomas Eiter and Heikki Mannila. Computing Discrete Frechet Distance. Technical report, Technishe Universität Wien, 1994.

[54] A. B. El-Rayes, Abd El-Moneim A. Mohamed, and Hamdy M. Abou Gabal. Linear Search for a Brownian target motion. *Acta Mathematica Scientia*, 23(3) :321–327, 2003.

[55] Yehuda Elmaliach, Noa Agmon, and Gal A Kaminka. Multi-Robot Area Patrol under Frequency Constraints. *Annals of Mathematics and Artificial Intelligence*, 57 :293–320, 2009. doi : 10.1007/s10472-010-9193-y.

[56] Kutluhan Erol. *Hierarchical Task Network Planning : Formalization, Analysis, and Implementation*. PhD thesis, University of Maryland, 1996.

[57] Pooyan Fazli, Alireza Davoodi, Philippe Pasquier, and Alan K Mackworth. Fault-Tolerant Multi-Robot Area Coverage with Limited Visibility. In *ICRA Workshop : Search and Pursuit/Evasion in the Physical World*, 2010.

[58] Silvia Ferrari, Rafael Fierro, and Domagoj Tolic. A Geometric Optimization Approach to Tracking Maneuvering Targets Using a Heterogeneous Mobile Sensor Network. In *IEEE Conference on Decision and Control, held jointly with the Chinese Control Conference (CDC-CCC)*, pages 1080–1087, 2009. ISBN 9781424438723.

[59] Eduardo Feo Flushing, Luca Gambardella, and Gianni A Di Caro. GIS-based Mission Support System for Wilderness Search and Rescue with Heterogeneous Agents. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop on Robots and Sensors integration in future rescue INformation system (ROSIN)*, 2012.

[60] Seng Keat Gan, Robert Fitch, and Salah Sukkarieh. Real-time decentralized search with inter-agent collision avoidance. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 504–510, May 2012. ISBN 978-1-4673-1405-3. doi : 10.1109/ICRA.2012.6224975.

[61] Brian P Gerkey, Sebastian Thrun, and Geoff Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research (IJRR)*, 1(c) :20–27, 2006.

[62] Subir K. Ghosh. Approximation algorithms for art gallery problems in polygons. In *Conference on Discrete Mathematics*, 2009.

[63] Arnaud Glad, Olivier Simonin, Olivier Buffet, and François Charpillet. Theoretical Study of Ant-based Algorithms for Multi-Agent Patrolling. *Proceedings - 2008 European Conference on Artificial Intelligence*, 2008.

[64] Arnaud Glad, Olivier Simonin, Olivier Buffet, and François Charpillet. Influence of Different Execution Models on Patrolling Ant Behaviors : from Agents to Robots. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[65] Hector González-Baños and Jean-Claude Latombe. A randomized art-gallery algorithm for sensor placement. In *7th annual symposium on Computational Geometry*, 2001. ISBN 158113357X.

[66] Ben Grocholsky, Rahul Swaminathan, James Keller, Vijay Kumar, and George Pappas. Information Driven Coordinated Air-Ground Proactive Sensing. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2223–2228, April 2005. ISBN 078038914X.

[67] JM Hereford and MA Siebold. Bio-inspired search strategies for robot swarms. *Swarm Robotics, From Biology to Robotics*, 2010.

[68] Joao P Hespanha, Hyoun Jin Kim, and Shankar Sastry. Multiple-agent probabilistic pursuit-evasion games. In *IEEE Conference on Decision and Control (CDC)*, 1999.

[69] G. Hollinger, S. Singh, J. Djugash, and a. Kehagias. Efficient Multi-robot Search for a Moving Target. *International Journal of Robotics Research (IJRR)*, 28(2) :201–219, February 2009. ISSN 0278-3649. doi : 10.1177/0278364908099853.

[70] Geoffrey Hollinger and Sanjiv Singh. Multi-robot coordination with periodic connectivity. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[71] Geoffrey Hollinger, Athanasios Kehagias, and Sanjiv Singh. GSST : anytime guaranteed search. *Autonomous Robots*, 29(1) :99–118, April 2010. ISSN 0929-5593. doi : 10.1007/s10514-010-9189-9.

[72] Geoffrey A. Hollinger and Sanjiv Singh. Multirobot Coordination With Periodic Connectivity : Theory and Experiments. *IEEE Transactions on Robotics*, 28(4) :967–973, August 2012. ISSN 1552-3098. doi : 10.1109/TRO.2012.2190178.

[73] G. Infantes, C. Lesire, and C. Pralet. Multi-Robot Planning and Execution for an Exploration Mission : a Case Study. In *ICAPS 2014 PlanRob Workshop*, 2014.

[74] Volkan Isler, Sanjeev Khanna, John Spletzer, and Camillo J Taylor. Target Tracking with Distributed Sensors : The Focus of Attention Problem. *Computer Vision and Image Understanding*, pages 1–37, 2005.

[75] Moshe Israel, Evgeny Khmelnitsky, and Evgeny Kagan. Search for a mobile target by ground vehicle on a topographic terrain. In *2012 IEEE 27th Convention of Electrical and Electronics Engineers in Israel*, November 2012. ISBN 978-1-4673-4681-8. doi : 10.1109/EEEI.2012.6377123.

[76] Adam Jacoff, Elena Messina, Brian A. Weiss, Satoshi Tadokoro, and Yuki Nakagawa. Test arenas and performance metrics for urban search and rescue robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3396–3403, 2003.

[77] James S. Jennings, Greg Whelan, and William F Evans. Cooperative Search and Rescue with a Team of Mobile Robots. In *IEEE International Conference on Advanced Robotics (ICAR)*, pages 193–200, 1997.

[78] Boyoon Jung and Gaurav S Sukhatme. Tracking Targets Using Multiple Robots : The Effect of Environment Occlusion. *Autonomous Robots*, pages 191–205, 2002.

[79] Nidhi Kalra, Dave Ferguson, and Anthony Stentz. Hoplites : A market-based framework for planned tight coordination in multirobot teams. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[80] Seema Kamath, Eric Meisner, and Volkan Isler. Triangulation based multi target tracking with mobile sensor networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[81] S Karaman and E Frazzoli. Incremental sampling-based algorithms for a class of pursuit-evasion games. *Algorithmic Foundations of Robotics IX*, 2011.

[82] Nikhil Karnad and Volkan Isler. Lion and man game in the presence of a circular obstacle. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5045–5050, October 2009. ISBN 978-1-4244-3803-7. doi : 10.1109/IROS.2009.5354443.

[83] Max Katsev, Anna Yershova, Benjamin Tovar, Robert Ghrist, and Steven M LaValle. Mapping and Pursuit-Evasion Strategies For a Simple Wall-Following Robot. *IEEE Transactions on Robotics*, 27(1) :113–128, February 2011. ISSN 1552-3098. doi : 10.1109/TRO.2010.2095570.

[84] Fotios Katsilieris, M Lindhé, and DV Dimarogonas. Demonstration of multi-robot search and secure. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[85] Stephen Kloder and Seth Hutchinson. Barrier coverage for variable bounded-range line-of-sight guards. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 10–14, 2007. ISBN 1424406021.

[86] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154, 2004.

[87] A. Kolling and S. Carpin. Extracting surveillance graphs from robot maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2323–2328, September 2008. ISBN 978-1-4244-2057-5. doi : 10.1109/IROS.2008.4650763.

[88] A Kolling, A Kleiner, M. Lewis, and K. Sycara. Pursuit-Evasion in 2.5D based on Team-Visibility. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[89] Andreas Kolling and Stefano Carpin. Cooperative Observation of Multiple Moving Targets : an algorithm and its formalization. *International Journal of Robotics Research (IJRR)*, 26 (9) :935–953, September 2007. ISSN 0278-3649. doi : 10.1177/0278364907080424.

[90] Andreas Kolling and Stefano Carpin. Pursuit-Evasion on Trees by Robot Teams. *IEEE Transactions on Robotics*, 26(1) :32–47, February 2010. ISSN 1552-3098. doi : 10.1109/TRO.2009.2035737.

[91] Andreas Kolling and Stefano Carpin. Multi-robot pursuit-evasion without maps. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[92] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP : Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. *Proceedings of Robotics : Science and Systems (RSS)*, 2008.

[93] AS LaPaugh. Recontamination does not help. *Princeton University, Computer Science Dept.*, 1982.

[94] Steven M LaValle, Hector H Gonzales-Banos, Craig Becker, and Jean-claude Latombe. Motion Strategies for Maintaining Visibility of a Moving Target. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 731–736, 1997.

[95] Adam N. Letchford, Saeideh D. Nasiri, and Dirk Oliver Theis. Compact formulations of the Steiner Traveling Salesman Problem and related problems. *European Journal of Operational Research*, 228 :83–92, 2013. ISSN 03772217. doi : 10.1016/j.ejor.2013.01.044.

[96] Jeffrey T. Linderoth and Andrea Lodi. MILP software. *Wiley encyclopedia of operations research and management science*, 2010.

[97] Ming Liu, Francis Colas, Luc Oth, and Roland Siegwart. Incremental topological segmentation for semi-structured environments using discretized GVG. *Autonomous Robots*, 38 (2) :143–160, 2015. ISSN 0929-5593. doi : 10.1007/s10514-014-9398-8.

[98] Marco Mamei and Franco Zambonelli. Spreading pheromones in everyday environments through RFID technology. In *IEEE Swarm Intelligence Symposium (SIS)*, pages 281–288, 2005.

[99] Stefan Markov and Stefano Carpin. A cooperative distributed approach to target motion control in multirobot observation of multiple targets. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[100] N. Megiddo, S. L. HAKIMI, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou. The Complexity of Searching a Graph. *Journal ofthe Association for Computing Machinery*, 1988.

[101] Bernhard Meindl and Matthias Templ. Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 2012.

[102] Yun-Qian Miao. *A study of mobility models in mobile surveillance systems*. PhD thesis, University of Waterloo, 2010.

[103] Yun-Qian Miao, Alaa Khamis, and Mohamed S. Kamel. Applying anti-flocking model in mobile surveillance systems. In *Autonomous and Intelligent Systems (AIS)*, June 2010. ISBN 978-1-4244-7104-1. doi : 10.1109/AIS.2010.5547036.

[104] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM*, 7(4) :326–329, October 1960. ISSN 00045411. doi : 10.1145/321043.321046.

[105] Faraz M. Mirzaei, Anastasios I. Mourikis, and Stergios I. Roumeliotis. On the performance of multi-robot target tracking. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[106] Mark Moors, T Rohling, and Dirk Schulz. A probabilistic approach to coordinated multi-robot indoor surveillance. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[107] Mark B. Moseley, Benjamin P. Grocholsky, Carol Cheung, and Sanjiv Singh. Integrated long-range UAV/UGV collaborative target tracking. *SPIE Defense, Security, and Sensing*, 2009.

[108] Alejandro R Mosteo. *Multi-Robot Task Allocation for Service Robotics : from Unlimited to Limited Communication Range*. PhD thesis, Universidad de Zaragoza, 2010.

[109] Roozbeh Mottaghi and Richard Vaughan. An integrated particle filter and potential field method applied to cooperative multi-robot target tracking. *Autonomous Robots*, 23(1) : 19–35, 2007.

[110] Richard M. Murray. Recent Research in Cooperative Control of Multivehicle Systems. *Journal of Dynamic Systems, Measurement, and Control*, 129 :571, 2007. ISSN 00220434. doi : 10.1115/1.2766721.

[111] R. Murrieta, A. Sarmiento, S. Bhattacharya, and S. Hutchinson. Maintaining visibility of a moving target at a fixed distance : the case of observer bounded speed. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 479–484 Vol.1, 2004. ISBN 0-7803-8232-3. doi : 10.1109/ROBOT.2004.1307195.

[112] R. Murrieta-Cid, B. Tovar, and S. Hutchinson. A sampling-based motion planning approach to maintain visibility of unpredictable targets. *Autonomous Robots*, 19 :285–300, 2005.

[113] R. Murrieta-Cid, T. Muppirala, Alejandro Sarmiento, Sourabh Bhattacharya, and Seth Hutchinson. Surveillance Strategies for a Pursuer with Finite Sensor Range. *International Journal of Robotics Research (IJRR)*, 26(3) :233–253, March 2007. ISSN 0278-3649. doi : 10.1177/0278364907077083.

[114] R. Murrieta-Cid, R. Sarmiento, S. Bhattacharya, and S. Hutchinson. Surveillance strategies for a pursuer with finite sensor range. *International Journal of Robotics Research (IJRR)*, 2007.

[115] R. Murrieta-Cid, R. Monroy, S. Hutchinson, and J.-P. Laumond. A Complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2657 –2664, may 2008.

[116] R Murrieta-Cid, Raul Monroy, Seth Hutchinson, and Jea. A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[117] Rafael Murrieta-Cid, Hector H Gonzales-Banos, and Benjamin Tovar. A reactive motion planner to maintain visibility of unpredictable targets. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[118] Rafael Murrieta-Cid, Ubaldo Ruiz, Jose Luis Marroquin, Jean-Paul Laumond, and Seth Hutchinson. Tracking an omnidirectional evader with a differential drive robot. *Autonomous Robots*, 31(4) :345–366, August 2011. ISSN 0929-5593. doi : 10.1007/s10514-011-9246-z.

[119] Paul J. Nahin. *Chases and Escapes : The Mathematics of Pursuit and Evasion (New in Paper)*. Princeton University Press, 2012.

[120] Eric Nettleton. *Decentralised architectures for tracking and navigation with multiple flight vehicles*. PhD thesis, University of Sydney, 2003.

[121] N. Noori and V. Isler. Lion and man with visibility in monotone polygons. *International Journal of Robotics Research (IJRR)*, 33(1) :155–181, September 2014. ISSN 0278-3649. doi : 10.1177/0278364913498291.

[122] Akira Ohsumi. Optimal search for a Markovian target. *Naval Research Logistics*, 38 (4) :531–554, August 1991. ISSN 0894069X. doi : {10.1002/1520-6750(199108)38:4⟨531::AID-NAV3220380407⟩3.0.CO;2-L}.

[123] J O'rourke. *Art gallery theorems and algorithms*. Princeton University Press, 1987.

[124] Thomas Oskam, Robert W. Sumner, Nils Thuerey, and Markus Gross. Visibility transition planning for dynamic camera control. In *SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, volume 1, New York, New York, USA, 2009. ACM Press. ISBN 9781605586106. doi : 10.1145/1599470.1599478.

[125] Mark Owen, Huili Yu, Tim McLain, and Randy Beard. Moving ground target tracking in urban terrain using air/ground vehicles. *Globecom Workshops*, pages 1816–1820, December 2010. doi : 10.1109/GLOCOMW.2010.5700254.

[126] Nikolaos P. Papanikolopoulos, Pradeep K. Khosla, and Takeo Kanade. Visual tracking of a moving target by a camera mounted on a robot : A combination of control and vision. *IEEE Transactions on Robotics and Automation*, 9(1) :14–35, 1993.

[127] Lynne E Parker. Distributed Algorithms for Multi-Robot Observation of Multiple Moving Targets. *Autonomous Robots*, pages 231–255, 2002.

[128] Lynne E. Parker and Brad A. Emmons. Cooperative Multi-Robot Observation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2082–2089, April 1997. ISBN 0780336127.

[129] TD Parsons. Pursuit-evasion in a graph. *Theory and applications of graphs*, 1978.

[130] Fabio Pasqualetti, Antonio Franchi, and Francesco Bullo. On optimal cooperative patrolling. In *IEEE Conference on Decision and Control (CDC)*, pages 7153–7158, 2010. ISBN 9781424477449.

[131] Gábor Pataki. Teaching Integer Programming Formulations Using the Traveling Salesman Problem. *SIAM Review*, 45(1) :116–123, 2003. ISSN 0036-1445. doi : 10.1137/S00361445023685.

[132] Nikos Pelekis, Ioannis Kopanakis, Gerasimos Marketos, Irene Ntoutsi, Gennady Andrienko, and Yannis Theodoridis. Similarity Search in Trajectory Databases. In *International Symposium on Temporal Representation and Reasoning (TIME)*, pages 129–140, 2007. ISBN 0-7695-2836-8. doi : 10.1109/TIME.2007.59.

[133] Luciano CA Pimenta, Mac Schwager, Quentin Lindsey, Vijay Kumar, Daniela Rus, Renato C Mesquita, and Guilherme AS Pereira. Simultaneous coverage and tracking (SCAT) of moving targets with robot networks. In *Algorithmic Foundation of Robotics VIII*, Springer Tracts in Advanced Robotics, pages 85–99. Springer Berlin / Heidelberg, 2010.

[134] Paolo Pirjanian and Maja Mataric. Multi-Robot Target Acquisition using Multiple Objective Behavior Coordination. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2696–2702, April 2000. ISBN 0780358864.

[135] James Pita, Manish Jain, F Ordónez, and Christopher Portway. Using game theory for Los Angeles airport security. *AI Magazine*, pages 43–57, 2009.

[136] David Portugal and Rui Rocha. MSP algorithm : multi-robot patrolling based on territory allocation using balanced graph partitioning. In *Symposium on Applied Computing*, pages 1271–1276, 2010. ISBN 9781605586380.

[137] David Portugal and Rui Rocha. A survey on multi-robot patrolling algorithms. *Technological Innovation for Sustainability*, pages 139–146, 2011.

[138] David Portugal and Rui P. Rocha. Distributed multi-robot patrol : A scalable and fault-tolerant framework. *Robotics and Autonomous Systems*, July 2013. ISSN 09218890. doi : 10.1016/j.robot.2013.06.011.

[139] Cyril Poulet. *Coordination dans les systèmes multi-agents : Le problème de la patrouille en système ouvert*. PhD thesis, Université Pierre et Marie Curie - Paris 6, 2013.

[140] Jim Pugh and Alcherio Martinoli. Inspiring and Modeling Multi-Robot Search with Particle Swarm Optimization. In *IEEE Swarm Intelligence Symposium (SIS)*, pages 332–339, April 2007. ISBN 1-4244-0708-7. doi : 10.1109/SIS.2007.367956.

[141] Kun Qian, X Ma, and X Dai. Simultaneous robot localization and person tracking using Rao-Blackwellised particle filters with multi-modal sensors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 22–26, 2008. ISBN 9781424420582.

[142] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS : an open-source Robot Operating System. *ICRA Workshop on open source software*, 3(3.2), 2009.

[143] Eric Raboin, Dana Nau, Ugur Kuter, Satyandra K Gupta, and Petr Svec. Strategy generation in multi-agent imperfect-information pursuit games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010.

[144] Eric Raboin, Ugur Kuter, and Dana S Nau. Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1201–1202, 2012.

[145] Rasmus Rasmussen. Tsp in spreadsheets - a fast and flexible tool. *Omega*, 39(1) :51–63, 2011.

[146] Alessandro Renzaglia. *Adaptive stochastic optimization for cooperative coverage with a swarm of Micro Aerial Vehicles*. PhD thesis, Université de Grenoble, 2012.

[147] James R. Riehl, Gaemus E. Collins, and Joao P. Hespanha. Cooperative graph-based model predictive search. In *IEEE Conference on Decision and Control (CDC)*, pages 2998–3004, 2007.

[148] Cyril Robin and Simon Lacroix. Failure anticipation in pursuit-evasion. In *Proceedings of Robotics : Science and Systems (RSS)*, 2012.

[149] Cyril Robin and Simon Lacroix. Anticipating failures in realistic pursuit-evasion. In *14ème Congrès des doctorants EDSYS*, 2013.

[150] Cyril Robin and Simon Lacroix. Multi-robot Target Detection and Tracking : Taxonomy and Survey. Technical report, LAAS/CNRS, 2014.

[151] Cyril Robin and Simon Lacroix. Taxonomy on Multi-robot Target Detection and Tracking. In *European Conference on Artificial Intelligence (ECAI), Workshop on Multi-Agent Coordination in Robotic Exploration (MACOREX)*, 2014.

[152] N Roy, GJ Gordon, and S Thrun. Finding approximate POMDP solutions through belief compression. *Journal of Artificial Intelligence Research (JAIR)*, 2005.

[153] Sui Ruan, Candra Meirina, Feili Yu, KR Pattipati, and RL Popp. Patrolling in a stochastic environment. Technical report, DTIC Document, 2005.

[154] Paul E Rybski, Sascha A Stoeter, Michael D Erickson, Maria Gini, Dean F Hougen, and Nikolaos Papanikolopoulos. A team of robotic agents for surveillance. In *Proceedings of the 4th International Conference on Autonomous Agents*, pages 9–16. ACM, 2000.

[155] Tiago Sak, Jacques Wainer, and Siome Klein Goldenstein. Probabilistic multiagent patrolling. *Advances in Artificial Intelligence*, pages 124–133, 2008.

[156] J Santos and P Lima. Multi-robot cooperative object localization. In *RoboCup 2009 : Robot Soccer World Cup XIII*, pages 332–343, 2010.

[157] Alejandro Sarmiento, Rafael Murrieta-Cid, and Seth Hutchinson. An Efficient Motion Strategy to Compute Expected-Time Locally Optimal Continuous Search Paths in Known Environments. *Advanced Robotics*, 23(12) :1533–1560, September 2009. ISSN 01691864. doi : 10.1163/016918609X12496339799170.

[158] Jacob T. Schwartz and Micha Sharir. *On the 'piano movers' problem*. New York University, Department of Computer Science, Courant Institute of Mathematical Sciences, 1982.

[159] Emilio Frazzoli Sertac Karaman, Tal Shima. Task assignment for complex uav operations using genetic algorithms. In *AIAA Guidance, Navigation, and Control Conference*, 2009.

[160] Kun Shi, Zhiqiang Cao, Wenwen Zhang, and Chao Zhou. The Targets Pursuit for Multi-robot System with Hybrid Wireless Sensor Networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 538–547, 2010.

[161] Trey Smith and Reid Simmons. Point-Based POMDP Algorithms : Improved Analysis and Implementation. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2012.

[162] D. E. Soltero, M. Schwager, and D. Rus. Decentralized path planning for coverage tasks using gradient descent adaptive control. *International Journal of Robotics Research (IJRR)*, October 2013. ISSN 0278-3649. doi : 10.1177/0278364913497241.

[163] Lawrence D. Stone. *Theory of Optimal Search (2nd Edition)*. Academic Press New York, 2007. doi : 10.2307/2286890.

[164] J Strom, R Morton, K Reilly, and E Olson. Online probabilistic pursuit of adversarial evaders. In *ICRA Workshop : Search and Pursuit/Evasion in the Physical World*, 2010.

[165] Zhijun Tang and U Ozguner. Motion planning for multi-target surveillance with mobile sensor agents. *IEEE Transactions on Robotics*, 21(5) :898–908, 2005.

[166] Herbert G Tanner. Switched UAV-UGV cooperation scheme for target detection. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[167] Panagiostis Theodorakopoulos. *On autonomous target tracking for UAVs*. PhD thesis, University of Toulouse, 2009.

[168] Nicolas a. Tsokas and Kostas J. Kyriakopoulos. Multi-robot multiple hypothesis tracking for pedestrian tracking. *Autonomous Robots*, 32(1) :63–79, November 2011. ISSN 0929-5593. doi : 10.1007/s10514-011-9259-7.

[169] Jorge Urrutia. Art gallery and illumination problems. *Handbook of computational geometry*, 2000.

[170] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem : A survey. *European Journal of Operational Research*, 209(1) :1–10, February 2011. ISSN 03772217. doi : 10.1016/j.ejor.2010.03.045.

[171] René Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. Probabilistic Pursuit–Evasion Games : Theory, Implementation, and Experimental Evaluation. *IEEE Transactions on Robotics and Automation*, 18(5) :662–669, 2002.

[172] MAM Vieira, Ramesh Govindan, and GS Sukhatme. Scalable and practical pursuit-evasion with networked robots. *Intelligent Service Robotics*, 2009.

[173] Christopher Vo and Jyh-ming Lien. Visibility-Based Strategies for Tracking and Searching Unpredictable Coherent Targets Among Known Obstacles. In *ICRA Workshop : Search and Pursuit/Evasion in the Physical World*, 2010.

[174] Zongyao Wang, Dongbing Gu, Tao Meng, and Yanzhi Zhao. Consensus Target Tracking in Multi-robot Systems. *Intelligent Robotics and Applications*, pages 724–735, 2010.

[175] El-mane Wong, Frédéric Bourgault, and Tomonari Furukawa. Multi-vehicle Bayesian Search for Multiple Lost Targets. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3180–3185, April 2005. ISBN 078038914X.

[176] Zhe Xu, B Douillard, P Morton, V Vlaskine, C Wohlleber, M Calleija, J.P. Underwood, and Salah Sukkarieh. Towards Collaborative Multi-MAV-UGV Teams for Target Tracking. In *Proceedings of Robotics : Science and Systems (RSS), Workshop on Integration of perception with control and navigation for resource-limited, highly dynamic, autonomous systems*, 2012.

[177] Zhe Xu, Robert Fitch, and Salah Sukkarieh. Decentralised Coordination of Mobile Robots for Target Tracking with Learnt Utility Models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2006–2012, 2013. ISBN 9781467356428.

[178] Huili Yu and RW Beard. Probabilistic path planning for cooperative target tracking using aerial and ground vehicles. In *American Control Conference (ACC)*, pages 4673–4678, 2011. ISBN 9781457700798.

[179] Ke Zhou and Stergios I Roumeliotis. Multirobot active target tracking with combinations of relative observations. *IEEE Transactions on Robotics*, 27(4) :678–695, 2011. doi : 10.1109/TRO.2011.2114734.

[180] Pu-Cheng Zhou, Bing-Rong Hong, Yue-Hai Wang, and Tong Zhou. Multi-agent coopoerative pursuit based on extended Contract Net Protocol. In *Machine Learning and Cybernetics*, August 2004. ISBN 0780384032.

RÉFÉRENCES DES ÉPIGRAPHES

[epi1] Box, George E.; Draper, Norman R. : *Empirical model-building and response surfaces*. John Wiley & Sons, 1987

[epi2] Char, René : *Fureur et Mystère*. 1948

[epi3] Dumas, Alexandre : *Les Trois Mousquetaires*. 1844

[epi4] Hopper, Grace M. : Interview of. In : *Information Week* (1987), March, S. 52

[epi5] Hugo, Victor : *Hernani*. Livre de Poche 2434, 1830. – 149 S

[epi6] Knuth, Donald E. : Computer Programming as an Art. In : *Communications of the ACM* 17 (1974), December, Nr. 12, S. 667–673

[epi7] Montesquieu : *Pensées diverses*

[epi8] Nimoy, Leonard : *A Lifetime Of Love : Poems On The Passages Of Life*. 2002

[epi9] Proudhon, Pierre-Joseph : *Les Confessions d'un révolutionnaire pour servir à l'histoire de la Révolution de Février*. Garnier frères, 1851

[epi10] Rostand, Jean : *Aux sources de la biologie*. Gallimard, 14ème édition, 1958

[epi11] Tzu, Sun : *L'Art de la guerre*

[epi12] Térence : *Le Phormion, v. 454 (Act II, sc. 4, l. 14)*

[epi13] Voltaire : *Lettres Philosophiques*. 1734

Sur la terre, tantôt sable, tantôt savane,
L'un à l'autre liés en longue caravane,
Echangeant leur pensée en confuses rumeurs,
Emmenant avec eux les lois, les faits, les mœurs,
Les esprits, voyageurs éternels, sont en marche.
L'un porte le drapeau, les autres portent l'arche ;
Ce saint voyage a nom Progrès. De temps en temps,
Ils s'arrêtent, rêveurs, attentifs, haletants,
Puis repartent. En route ! ils s'appellent, ils s'aident,
Ils vont ! Les horizons aux horizons succèdent,
Les plateaux aux plateaux, les sommets aux sommets.
On avance toujours, on n'arrive jamais.

— Victor Hugo
Les Châtiments, La Caravane

Ces travaux ont été partiellement subventionnés par la DGA
à travers une bourse de thèse (réf. DGA 2011-60-096) et le PEA ACTION

Ce document a été rédigé sous LaTeX
à partir du modèle `classicthesis` développé par André Miede.

Manuscrit de thèse de Cyril Robin– version 1.1 en date du 20 août 2015

DÉCLARATION

Je, soussigné Cyril J. R. Robin, certifie par la présente que les travaux présentés ici et le présent document sont originaux et forment le fruit de mes propres recherches.

I, Cyril J. R. Robin, hereby certify that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

Toulouse, le 6 juin 2015